НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр КОВАЛЬ
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Інструментальні засоби посиленого захисту закритих ресурсів»

Виконав (-ла):
студент (-ка) IV курсу, групи ТМ-62
Гусейнов Раміз Нематович _____

Керівник:
Професор, д.т.н, професор кафедри Апепс
Отрох Сергій Іванович _____

Рецензент:
Доцент, к.т.н., доцент кафедри Технічної Кібернетики
Зенів Ірина Онуфріївна _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр КОВАЛЬ
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Гусейнову Рамізу Нематовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Інструментальні засоби посиленого захисту закритих ресурсів»

керівник роботи_проф. Отрох Сергій Іванович_
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи _JavaScript, Angular, NodeJs, express, Heroku, HMAC___

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_розробити алгоритм генерації одноразового паролю, реалізувати код сервера автентифікації, реалізувати код додатку для генерації паролів, реалізувати додаток для використання системи, інтегрувати модулі один з одним.

5. Перелік ілюстративного матеріалу
«Постановка задачі», «Огляд існуючих систем», «Огляд підходів до розв’язання проблеми», «Використані технології», «Архітектура системи», «Інтерфейс модулів системи», «Огляд взаємодії системних модулів», «Кінцевий результат», «Висновки»
6. Дата видачі завдання ”_14_”___жовтня___2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	14.10.19	
2.	Вивчення та аналіз задачі	14.10.2019-23.12.2019	
3.	Розробка архітектури та загальної структури системи	02.02.2020-03.03.2020	
4.	Розробка структур окремих підсистем	04.03.2020-14.04.2020	
5.	Програмна реалізація системи	15.04.2020-19.05.2020	
6.	Оформлення пояснювальної записки	20.05.2020-03.06.2020	
7.	Захист програмного продукту	30.05.2020	
8.	Передзахист	9.06.20	
9.	Захист	19.06.20	

Студент _____ _____Раміз ГУСЕЙНОВ_____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _Сергій ОТРОХ_____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою роботи є розробка програмного забезпечення для інструментального захисту закритих ресурсів, яке являтиме собою систему автентифікації. Система підвищує безпеку закритого ресурсу надаючи додатковий фактор автентифікації і робить доступ до системи складнішим для зловмисників.

Застосунок буде корисний для підприємств, що мають внутрішні ресурси з веб доступом через мережу інтернет.

Обсяг пояснювальної записки – 87 аркушів, містить 24 ілюстрації, 7 таблицm та 3 додатки.

Ключові слова: автентифікація, безпека інформаційних систем.

ANNOTATION

The aim of the work is to develop software for protection of closed resources, which will be an authentication system. The system increases the security of the closed resource by providing an additional authentication factor and makes access to the system more difficult for attackers.

The application will be useful for companies that have internal resources with web access via the Internet.

Explanatory Note Volume - 87 pages, contains 24 illustrations, 7 tables and 3 applications.

Keywords: authentication, information systems security.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

БД – база даних

СКДБ – Система керування базами даних

HTTPS – HyperText Transfer Protocol Secure

TFA – Two Factor Authentication

MAC – Message authentication code

PaaS – Platform as a Service

OTP – One-time password

IDE– Integrated Development Environment

ВСТУП

Незважаючи на відомі проблеми безпеки з використанням статичних паролів, паролі все ще є найпопулярнішим методом аутентифікації кінцевих користувачів. Одним з найважливіших завдань для адміністраторів веб-ресурсів є комплекс технічних та організаційних заходів, які направлені спільною концепцією безпеки і дозволяють значно покращити рівень захищеності та запровадити реакції веб-порталів на загрози. Зі зростанням ролі інформаційно-комунікаційних технологій у сучасному соціумі проблема захисту даних від втртати, викрадення, спотворення або пошкодження інформації потребує посиленої уваги. Вирішення проблем захисту інформації сприяє забезпеченню інформаційної безпеки як окремої особистості, організації, так і держави в цілому. Інформаційна безпека - це захист систем передавання, опрацювання та зберігання даних, при якому зберігається конфіденційність та цілісність даних. Також інформаційна безпека включає в себе комплекс заходів, які спрямовані на забезпечення захисту даних від доступу несанкціонованими особами, їх використання, росповсюдження чи внесення несанкціонованих змін. Атаки грубою силою та перебір за словником на створені користувачем паролі часто можливі завдяки їх обмеженій ентропії. За даними Ashley Vance, 20% усіх існуючих паролів охоплені переліком лише 5000 комбінацій. Тому, задля підвищення безпеки частою вимогою є певні правила до формування пароля, вимагаючи, наприклад, певної мінімальної кількості символів, включення специфічних символів, окрім буквено-цифрових. Це часто створює небажаний конфлікт між безпекою та зручністю використання, як це підкреслюється в контексті вибору пароля, управління ними та його складу – досить часто через всі ці обмеження користувачі обирають самий простий пароль, який задовольняє такі вимоги. Багатофакторна аутентифікація з'явилася як альтернативний спосіб поліпшення безпеки, вимагаючи від користувача надати більше, ніж один фактор аутентифікації, на відміну від пароля.

Зазвичай використовуються такі фактори аутентифікації:

- Знання – те, що користувач знає, наприклад, пароль;
- Володіння – те, що користувач має, наприклад токен автентифікації (програмний або апаратний);
- Притаманність – те, чим користувач є, наприклад, біометрична характеристика (відбиток пальця).

Аутентифікація є одним із основних аспектів безпеки, оскільки вона підтверджує ідентичність особи та гарантує, що дані не були підмінені в процесі передачі. Коди аутентифікації повідомлень (MAC) є одним з найважливіших інструментів аутентифікації та цілісності даних. Під час проходження написання дипломної роботи ми покращимо криптографічні характеристики методу HMAC, що є типом MAC функцій, який використовує хеш-функції. На основі покращеного алгоритму буде створена система авторизації користувачів.

1. ПОСТАНОВКА ЗАДАЧІ

Ми стикаємося з аутентифікацією досить часто: на роботі, при перегляді сайтів, при користуванні своїм смартфоном, коли ми використовуємо ім’я користувача та пароль для входу до комп’ютера чи веб-сайта, ми використовуємо один фактор – це знання пароля. Коли ми вводимо PIN-код і знімаємо гроші з банкомату, ми використовуємо два фактори – володіння картою банку і знання пін коду. Багато людей не виходять за межі використання цих двох факторів у своєму повсякденному житті. Для тих із нас, хто має доступ до більш захищених об'єктів, таких як центри обробки даних, фінансові або військові установи, ми можемо побачити більше розвинутих методів аутентифікації.

Крадіжка даних для аутентифікації несе велику загрозу, як для компаній, які можуть зазнати серйозних наслідків як з точки зору фінансів, так і репутації, а також для користувачів, які можуть бути причетними до незаконних дій, вчинених атакуючою особою. Хоча надійні системи аутентифікації представляють значну стійкість до крадіжок особистих даних, прості системи "ім'я користувача / пароль" є дуже вразливими, особливо у випадках, коли користувачі (з низькою обізнаністю про ризики) вибирають прості паролі.

Однією з найважливіших умов розробки безпечної інформаційної системи є впровадження безпечного і надійного методу аутентифікації користувачів, який не дасть змоги зловмисникам отримати доступ до закритого ресурсу.

Двофакторна аутентифікація (TFA) все частіше використовується для захисту та ідентифікації користувача. Двофакторна аутентифікація працює як додатковий крок у процесі аутентифікації, другий рівень захисту, який дає змогу відрізнити користувача від зловмисника. Зі збільшенням кіберзлочинів з кожним роком все більше і більше підприємств (починаючи від фінансових установ до роздрібної торгівлі (Amazon, eBay, Apple) впроваджують TFA як спосіб забезпечення довіри користувачів до своїх систем, що, в свою чергу, зменшує ризик будь-яких зловмисних користувачів від проникнення в їхні системи. Одними з основних алгоритмів двофакторної авторизації є HOTP (HMAC-Based

One-Time Password Algorithm) та TOTP (Time-Based One-Time Password Algorithm).

Метою розробки буде впровадження системи двофакторної аутентифікації, яка унеможливлює для злочинника доступ до ресурсів системи, а також надасть час для впровадження відповідних дій в протидії несанкціонованому доступу. Система буде працювати на одноразових паролях, які будуть формуватися на основі TOTP з використанням покращеного HMAC[1] алгоритму.

2.АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ПОСИЛЕНОГО ЗАХИСТУ ЗАКРИТИХ РЕСУРСІВ

При використанні двофакторної аутентифікації доступ до облікового запису можливий тільки з довірених пристроїв. При першому вході на новому пристрої вам буде потрібно надати два види інформації: ваш пароль і цифровий код підтвердження, який автоматично відображається на довірених пристроях. Після введення коду новий пристрій включається в число довірених пристроїв.

Оскільки для доступу до облікового запису при двофакторній ідентифікації недостатньо тільки знання пароля, безпека ваших даних істотно зростає.

2.1 Способи захисту закритих ресурсів

Загальнодоступні веб-додатки можуть бути використині хакерами як обчислювальні ресурси або інструменти заробітку. Інформація отримана в результаті злому може бути використана різними способами: надання платного доступу несанкціонованим користувачам, побудова бот-мережах і т.д. Особа власника інформації не важлива, оскільки процес злому автоматизований і поставлений на потік. Вартість іфнормації пропорційна потенційній вигоді, яку можна з неї отримати, а також фінансовими втратами підприємства, яке володіє інформацією.

Перш ніж шукати уразливості вручну необхідно перевірити додаток автоматизованими засобами. Такі засоби виконують цілий комлпекс перевірок на загальновідомі вразливості, наприклад, за допомогою SQL- ін’єкції.

Існуючі інструменти використовують різні способи захисту персональних даних та закритих ресурсів, наприклад інструмент OpenVAS сканує вузли мережі на наявність вразливостей і дозволяє управляти ними, інструмент OWASP Xenotix XSS Exploit Framework виконує перевірку ресурсі на наявність XSS-

вразливостей, інструмент Approof перевіряє конфігурацію веб-додатків, сканує на наявність вразливих компонентів, незахищених чутливих даних і шкідливого коду. Також існує багато онлайн сервісів які забезпечують захист ресурсів, найпоширенішими з них є SecurityHeaders.io – здійснює перевірку на наявність і коректність HTTP заголовків сервера, які призначені для забезпечення безпечного обміну даними, Observatory by Mozilla також надає змогу перевірити наявність проблем з безпекою для веб ресурсу (Слід зазначити, що окрім своїх результатів, при виборі відповідної опції, даний інструмент збирає і додає до звіту аналітику із посторонніх сервісів призначених для аналізу захищеності), One button scan здійснює сканування різних компонентів ресурсу на наявність вразливостей: DNS-сервер, коректність HTTP-заголовків, перевірка наявності SSL, перевірка відсутності чутливих даних. CSP Evaluator виконує перевірку на дотримання політики безпеки вмісту (CSP) і стійкість до XSS атак, SSL Server Test здійснює аналіз коректної конфігурації SSL для веб-сервера, ASafaWeb здійснює перевірку на відсутність загальновідомих вразливостей конфігурації сайтів, написаних на ASP.NET, Snyk надає змогу сканувати додатки написані на JavaScript, Ruby та Java на наявність певних вразливостей і, за необхідністю, виправляє певні проблеми безпеки.

Перед скануванням веб-додатку онлайн-сервісами необхідно ознайомитися з умовами користування для додатку. Деякі з систем сканування на вразливості публікують загальнодоступні звіти про перевірені сайти. Результати автоматизованого тестування показують різноматінті види потенційних загроз. По списку знайдених загроз формується пояснення і способи уникнення данихвразливостей, в першу чергу слід звертати увагу на критичні зауваження.

Після того, як в додаток внесені рекомендовані зміни по безпеці, потрібно просканувати повторно, щоб переконатися в правильності прийнятих заходів.

Одним із способів захисту закритих ресурсів є захист призначених для користувача даних за допомогою HTTPS. HyperText Transfer Protocol Secure

(HTTPS) є розширенням HTTP, яке використовує шифрування для передачі даних кінцевих користувачів, сучасні веб браузери інколи зовсім забороняють доступ до веб ресурсу, якщо він не використовує HTTPS. HTTPS гарантує конфіденційність та цілісніть даних при взаємодії з сервером. Популярність HTTPS є дуже високою і незабаром його використання буде невід'ємною складовою захисту даних. Використання HTTPS виключно необхідне, якщо користувачі передають на сервер будь які чутливі дані: інформацію про платіжні картуки персональну інформаці та навіть адреси відвіданих сторінок. Якщо при відправці даних з форми авторизації встановлюються cookie заголовки, які потім з кожним запитом насдилаються до сервера, зловмисник може отримати і використати для злочинних цілей. В результаті він отримує доступ до даних для сесії і отримує доступ до ресурсу в ролі, яка була надана жертві. Для того щоб запобігти подібним ситуаціям всі комунікації між сервером і користувачем мають проходити через HTTPS на всіх сторінках сайту[2]. Якщо HTTPS вже налагоджений, додатково можна використати HTTP Strict Transport Security (HSTS) – який є HTTP хедером, слугує для заборони використання використання незахищеного з'єднання для цього домену.

Запобігання міжсайтового скриптинга також є одним із способів захисту закритих ресурсів. Міжсайтовий скриптинг (XSS) – тип атаки на веб-ресурси, який полягає у ін'єкції на сторінку порталу постороннього коду, який виконується на браузері користувача, може змінювати сторінку і передавати викрадену інформацію зловмисникам.

Наприклад, якщо на сторінці наявна секція з коментарями для якої відсутня валідація вводу-виводу, зловмисник може ввести коментар, який буде містити шкідливий JavaScript код. В результаті у користувачів, які відкривають сторінку з коментарями, буде виконано шкідливий код і інформація про сесію користувача може бути передана зловмисникам. Особливо схильні до цього різновиду вразлиовсті сучасні веб-додатки, де сторінки побудовані з призначеного для користувача контенту, фронтенд-фреймворками, що інтерпретуються, наприклад Angular і Ember. Всі ці фреймворки вбудований

захист від XSS, але змішана генерація контенту на стороні сервера і клієнта створює нові види атак: впровадження директив Angular та хелперів Ember.

Захист від подібного типу вразливостей вимагає концентрації на контенті призначеному для користувача, для того щоб уникнути некоректної інтерпретації браузером. Цей захист чимось подібний до захисту від від SQL-ін’єкцій. При динамічній генерації HTML сторінок необхідно використовувати спеціальний інструментарій фреймворку, який вміє автоматично уникати даних вразливостей а також шаблонізатори, які виконують екранізацію спеціальних символів автоматично.

Політика безпеки вмісту (CSP) – це ще один інструмент захисту від XSS-атак. CSP – це спеціальні заголовки сервера, які визначають білий список джерел, які браузер може використовувати для завантаження даних для різних типів ресурсів. Наприклад, такі заголовки можуть заборонити запускати на домені скрипт зі стороннього домена. Завдяки цим політикам, браузер не буде виконувати код зі сторонніх ресурсів навіть, якщо він був впроваджений.

2.2 Апаратні засоби захисту

Для захисту інформації на рівні апаратного забезпечення використовуються апаратні ключі та засоби блокування пристроїв і інтерфейсів вводу-виводу інформації. Апаратні засоби захисту інформації – це різноманітні за принципом дії, побудовою і можливостями технічні конструкції, що забезпечують припинення розголошення, захист від витоку і протидію несанкціонованому доступу до джерел конфіденційної інформації; пристрої, вбудовані в блоки інформаційної системи (сервера, комп'ютери і т.д.). Вони призначені для внутрішнього захисту елементів обчислювальної техніки та засобів комунікації обчислювальних систем.

Коли говорять про захист даних, то мають на увазі дві основних проблеми втрати даних: підміни даних та несанкціонованого доступу до них. Боротьба з обома небезпеками ведеться різноманітими апаратними та програмними засобами а також комплексом організаційних заходів. До

апаратних засобів захисту від пошкодження даних слід віднести використання джерел неперервного живлення, а також резервування та архівування даних, засобів апаратного шифрування даних.

При видачі USB карти користувачу вона реєструється в RSA менеджері авторизації, карта зберігає в собі приватний ключ і має внутрішній годинник. Зазвичай такі коди є валідними якийсь певний фіксований проміжок часу. На основі ключа і часової відмітки, яка має бути синхронізованою з відміткою сервера аутентифікації, за певним алгоритмом генерується токен, який при авторизації до системи надсилається до менеджера авторизації, там використовуючи той самий приватний ключ і часову мітку генерується токен, якщо токени збігаються – то аутентифікація успішна. Приклад системи двофакторної авторизації на OTP (одноразових паролів).

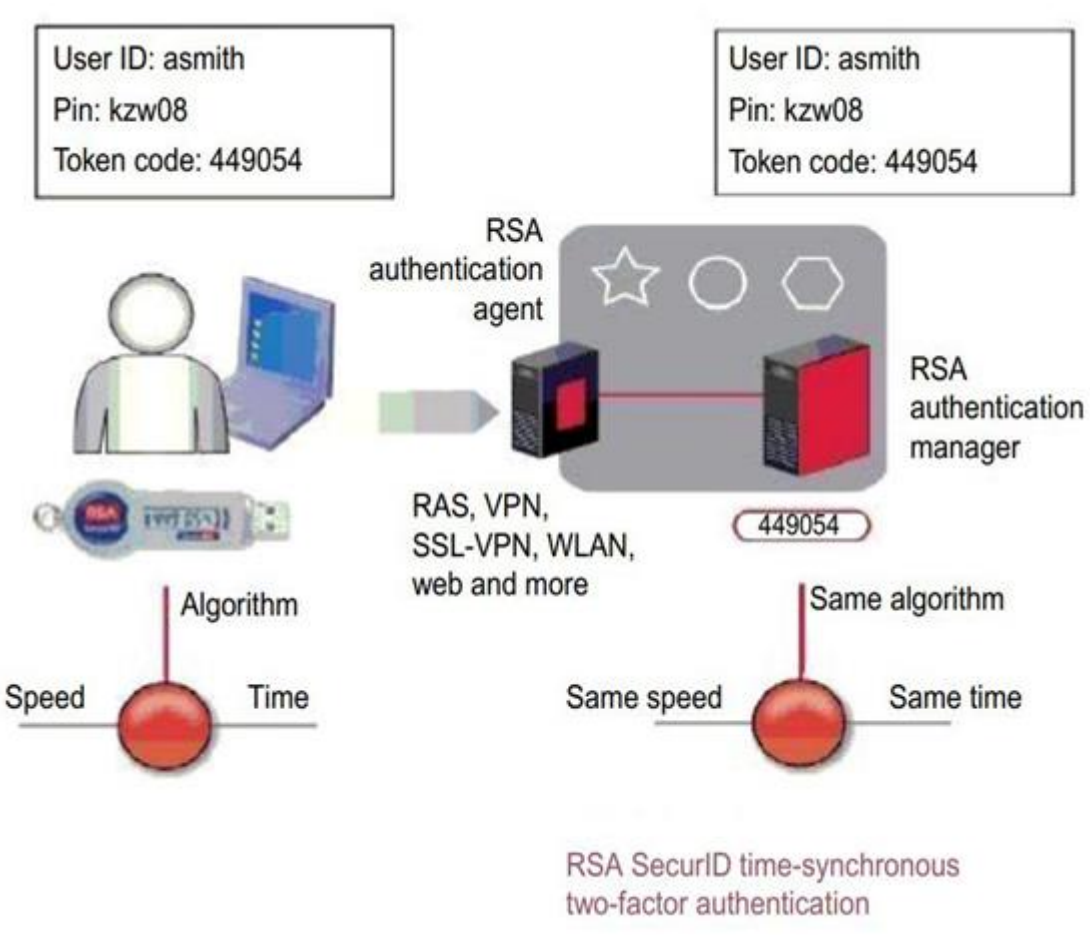

Рисунок 2.1 — Система RSA SecurID система аутентифікації

Іншим варіантом безконтактних токенів є асинхронні токени в яких замість часу синхронізованого між токеном і сервером використовується схема виклик-відповідь, під викликом розуміється випадкова кодова фраза яка надсилається до користувача за допомогою іншого каналу зв’язку ніж система до якої він намагається отримати доступ, наприклад смс, або код надісланий до мобільного додатку, ввівши кодову фразу до токену токен за допомогою секретного ключа генерить токен автентифікації який потім верифікується автентифікаційним сервером з використанням спільного секретного ключа.

Рисунок 2.2 — Digipass 270 токен аутентифікації банку

Апаратні токени Yubico Nano і Neo розроблені для передачі згенерованих одноразових паролів за NFC (зв'язок на невеликих відстанях) або

USB (емуляція введення на клавіатурі). Ці пристрої дійсно роблять двофакторну автентифікацію набагато простішою для кінцевих користувачів; однак є ще деякі недоліки. Використовувати токен на основі USB на більшості мобільних пристроїв без додаткового обладнання неможливо, а діапазон активності токенів NFC обмежений 15 см.

Рисунок 2.3 — Yubico токен автентифікації

2.3 Програмні засоби захисту

Програмні токени — це програми, що працюють на обчислювальних приладах, як правило, мобільних телефонах[3]. Сучасні мобільні операційні системи дозволяють створювати складні та потужні мобільні додатки, тому програмні токени надають додаткові функції, такі як безліч профілів автентифікації, здатність зчитувати секретні ключі за допомогою камери мобільного телефону у вигляді QR-коду (двовимірний штрих-код) для зручного і швидкого початку роботи з системою автентифікації та хмарне резервне

копіювання. Однак основна функціональність програмних токенів (генерація OTP) підтримується навіть на моделях, не визначених як "смартфони".

Значно надійнішими є програмні засоби, які використовують криптографічні методи захисту з несиметричними алгоритмами. В цих засобах для кожного сеансу зв'язку автоматично розповсюджуються ключі для шифрування. Такі ключі дійсні лише протягом одного сеансу. Після завершення сеансу вони вилучаються. Функції розповсюдження і вилучення ключів виконують сертифікаційні центри, де реєструються всі користувачі. За умови використання таких засобів можливе також використання електронного підпису (генерування, надання і перевірка). Аутентифікація при використанні криптографічних методів з несиметричними алгоритмами може бути проста і сувора. При суворій аутентифікації використовуються два криптографічних ключі для кожного абонента.

Рисунок 2.4 — Додаток Mobile-OTP

MobileOTP (MOTP) був одним з перших програмних токенів, розроблених для двофакторної аутентифікації. Перша версія MOTP була опублікована в 2003 році і передбачалася для запуску в основному на звичайних телефонах з підтримкою Java (не смартфонами). Коди OTP, згенеровані MOTP, є буквено-цифровими кодами, згенерованими на основі хеша MD5 секретного ключа, поточної позначки часу та персонального ідентифікаційного номера (PIN), що вводиться кінцевим користувачем щоразу, коли потребує створення OTP. MOTP мав такий же рівень безпеки, як і ключі згенеровані на основі

апаратних засобів; однак він спочатку був розроблений для роботи на звичайних стільникових телефонах (зараз їх називають «звичайними»), і тому йому не вистачає деяких особливостей, особливо процесу реєстрації, розробленого для здійснення у напрямку «клієнт-сервер». Це означає, що унікальний ключ (секрет) потрібно створити на пристрої, а потім ввести в профіль користувача на сервері аутентифікації. На рис. 2.4 показані фотографії програм MOTP Java (MIDlets), що працюють на телефоні Nokia 3510.

Рисунок 2.5 — Додаток Google Authenticator

Google Authenticator (Рис. 2.5) — це мобільний додаток, який використовує алгоритми TOTP або HOTP, які описано у стандарті RFC 6238. Алгоритм генерації OTP на отриманні HMAC хешу від секретного ключа та лічильного значення (часова мітка у випадку TOTP). Процес реєстрації, який відрізняється від MOTP, здійснюється на сервері, і в більшості випадків він проходить у вигляді зчитування QR-коду за допомогою камери, якщо на пристрої немає камери, то використовується ручне введення.

Розглянемо різницю між Google Authenticator та Mobile OTP. Google Authenticator використовує алгоритм TOTP, який схожий на MOTP, але відрізняється від нього. У системах, що базуються на TOTP, ключ генерується на сервері та потім відображається клієнту в процесі реєстрації. Зокрема, у Google Authenticator ключ відображається як QR-код, який потрібно сканувати

програмою, що робить процес реєстрації надзвичайно простим. На наш погляд, це головна перевага Google Authenticator, і саме тому такі системи стають популярними. Однак є ще кілька ключових факторів, які відрізняються, як показано таблиці 2.1.

Таблиця 2.1 — Відмінності Google Authentificator від MobileOTP

	MobileOTP	**Google Authenticator (TOTP)**
OTP алгоритм	MD5	HMAC
Час роботи пароля	10 с	30 с
Додатковий ПІН захист	Так	НІ
Генерація ключа	На клієнті	На сервері
Швидка реєстрація (з QR)	НІ	Так
Використання RFC	Ні	Так

2.4 Висновки до розділу 2

Найчастіше програмні засоби захисту інформації застосовують для виконання таких процесів як ідентифікація й аутентифікація користувачів, розмежування доступу користувачів до інформаційної мережі, парольний захист і перевірка повноважень, шифрування інформації, а також її захист від несанкціонованих змін, зчитування, копіювання.

Найбільшу увагу розробники й користувачі сьогодні приділяють програмним засобам. Організаційні, технологічні й апаратні методи захисту, як правило, не можуть бути здійснені без програмної складової. При цьому слід мати на увазі, що вартість здійснення багатьох програмних системних рішень із захисту інформації суттєво перевищує за затратами апаратні, технологічні й

організаційні рішення. Іншими недоліками програмних засобів захисту інформації є використання ресурсів системи, що призводить до зниження її ефективності, принципова можливість обходу такого захисту або його злочинної зміни в процесі експлуатації.

3. ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТУ

Програмний засіб був розроблений з використанням можливостей мови JavaScript та системи керування базами даних MongoDB. Для реалізації програмного продукту також було використано фреймворк з відкритим кодом Angular та платформу NodeJS.

3.1 Платформа NodeJS для розробки бекенд застосунків мовою JavaScript

Для розробки програмного продукту було обрано платформу з відкритим кодом NodeJS яка використовується для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript (Рис. 3.1). Якщо раніше JavaScript міг застосовуватися лише для обробки даних в браузері користувача а також формування динамічного контенту, то платформа node.js надала можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їх виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою кількістю можливостей застосування. Платформу було використано для розробки серверної частини додатків а також менеджера автентифікації. Node.js є середовищем виконання JavaScript, яке побудовано на рушії V8. Node.js використовує подійно-орієнтовану модель з неблокуючим введенням/виведенням, що робить його легким і ефективним.

Платформа Node.js дає можливість писати сервери за допомогою JavaScript з неймовірною продуктивністю. Як сказано у документації: Node.js — середовище виконання, яке використовує той самий рушій V8 Javascript, який ви можете знайти в браузері Google Chrome.

Node.js має наступні властивості:

- Асинхронна однопоточна модель виконання запитів;

- Використання асинхронного ввід/вивід;
- Вбудована система модулів CommonJS;
- Рушій JavaScript Google V8;

За замовчуванням в якості пакетного менеджеру використовується npm (node package manager).

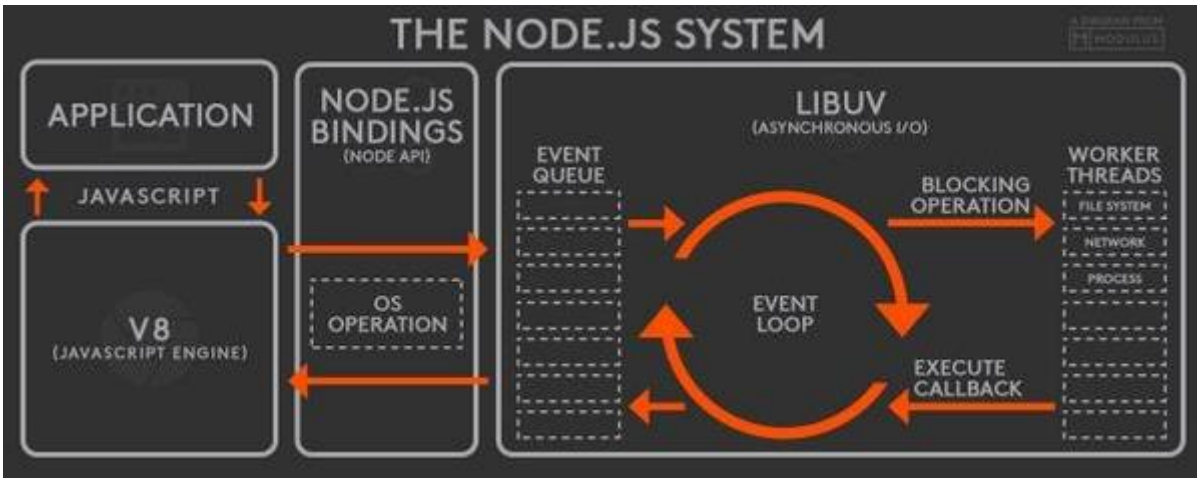

Рисунок 3.1 — Ілюстрація механізму асинхронної роботи NodeJS

Завдяки своїй високій популярності, мова JavaScript в наш час постійно розвивається, а сучасні процеси потерпіли суттєвих змін в порівнянні з недавнім минулим. Node.js — це серверна платформа, яка призначена для створення масштабованих розподілених серверних додатків, побудована на подієорієнтованій архітектурі в поєднанні з неблокуючою, асинхронною взаємодією. Головна ідея Node.js - це використання неблокуючого подієво-орієнтованого вводу / виводу, для того щоб використовувати невеликий обсяг обчислювальних ресурсів для взаємодії з додатками, що обробляють великі обсяги даних в реальному часі і функціонують на розподілених серверах. Все це означає, що Node.js не є універсальною платформою на всі випадки життя, а навпаки, використовується лише для вирішення завдань певного типу. Якщо додаток буде виконувати тяжкі обчислення, то варто відмовитися від використання Node.js, так як виконання тяжких обчислень будуть мати

негативний наслідок на продуктивність всього додатку через однопоточну природу платформи. Але за необхідності створення системи, яка з мінімальними ресурсами буде підтримувати багато одночасних http підключень, причому із двунаправленою взаємодією, то знадобиться організація асинхронного доступу.

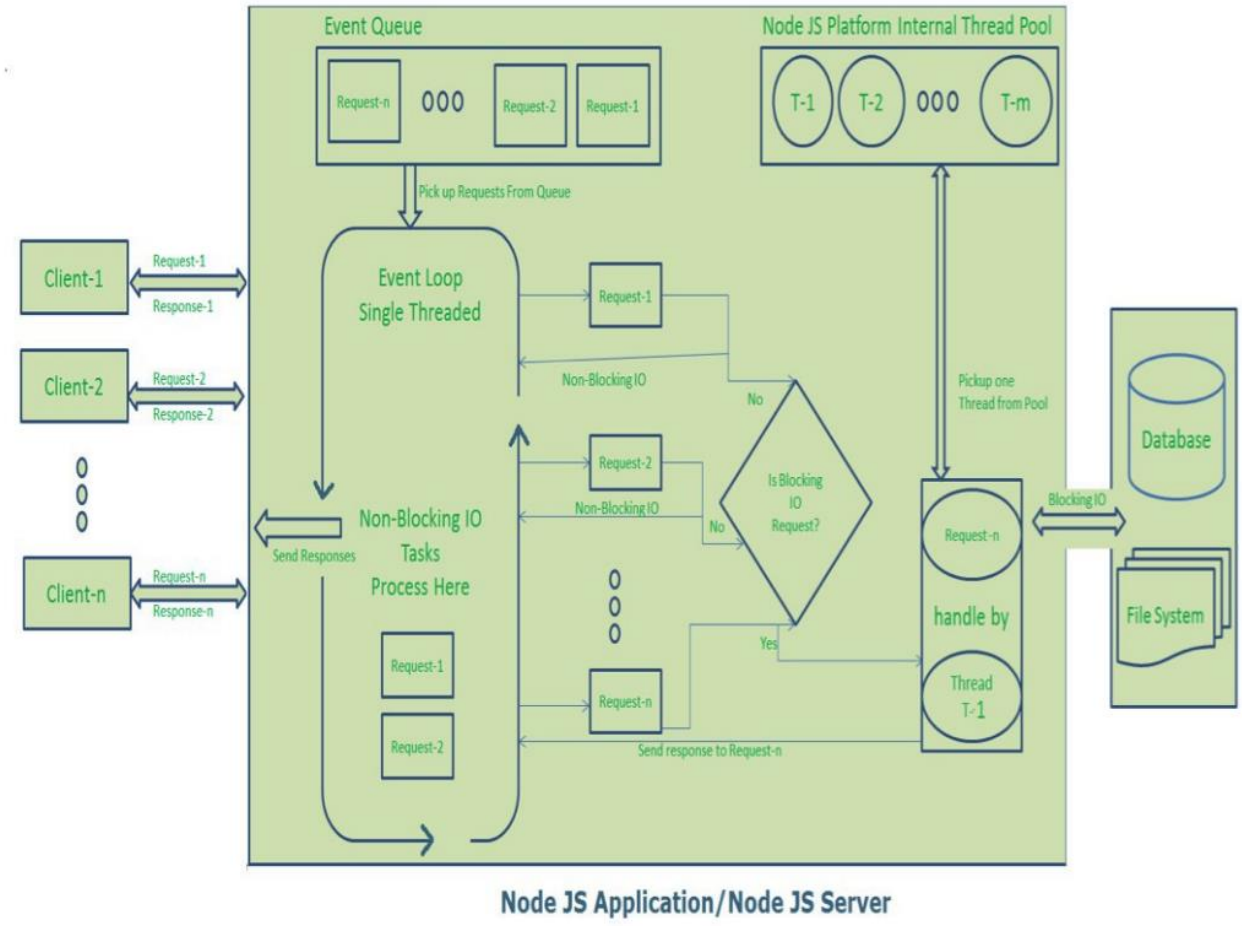

Рисунок 3.2 — Схематичне зображення серверу NodeJS

Асинхронна модель nodejs вирішує проблему одночасного доступу принципово по іншому. Вона побудована на циклічній черзі подій (eventloop). При виникненні певної події (прийшов http запит, результат запиту в бази даних, зчитування частини файлу) вона розміщується в кінець черги[4]. Потік виконання, що обробляє цей цикл, виконує код, пов'язаний з наступним подією, і, в свою чергу, розміщує його в кінець черги. Цей процес відбувається до тих

пір, поки черга містить певні події або функції, які чекають на настання певних подій. На даний момент ця технологія розвивається дуже швидко, та має підтримку глобальних компаній, таких як: HBO, PayPal, Netflix, Google, Microsoft, IBM та багатьох інших. В 2015 вийшла 4.0.0 версія платформи Node.js. IBM заявив про необхідність інтегрування функції Node.js в свою лінійку програмних продуктів для розробки корпоративних додатків як хмарних так і розміщених на серверах підприємства. Node.js — це платформа, що використовується здебільшого для вирішення проблем багатокористувацьких сервісів які не потребують важких обчислень. Вона успішно застосовується для розробки веб-чатів, браузерних ігор, інформаційних порталів. Здебільшого проблеми з використанням Node.js в разі використання плафторми не для тих задач, для яких вона була створена.

3.2 Фреймворк Angular для інтерфейсу користувача

Фреймворк Angular з відкритим сирцевим кодом написаний на TypeScript, основну розробку проводить Google та спільнота індивідуальних розробників та підприємств. Було використано для розробки додатку до якого застосовується двофакторна авторизація. А також на його основі побудований додаток для видачі ключів і генерації паролів кінцевими користувачами.

Бібліотека зазвичай спеціалізується на вузькій чи конкретній задачі. jQuery наприклад, спеціалізується на роботі з DOM-деревом, створює запити на backend тощо. Але бібліотека не передбачає створення архітектурної системи в цілому. Фреймворк, натомість, призначений для того, щоб побудувати архітектуру усього застосунку.

Перша версія Angular, яку сьогодні заведено називати Angular JS, була багато у чому не оптимальною. Однак величезна кількість проектів і досі використовують цю версію. Згодом команда розробників фактично з нуля переписала проект. Нова версія отримала назву Angular 2 чи просто Angular без

індексу. Вона кардинально відрізнялась від першої, справляючи враження зовсім іншого фреймворку. Проекти не могли з легкістю мігрувати зі старої версії на нову, тому це зумовило певні проблеми. Необхідно було переписати усе повторно. Angular став концептуально простішим, більш технологічним та зрозумілішим (див. рисунок 3.3).

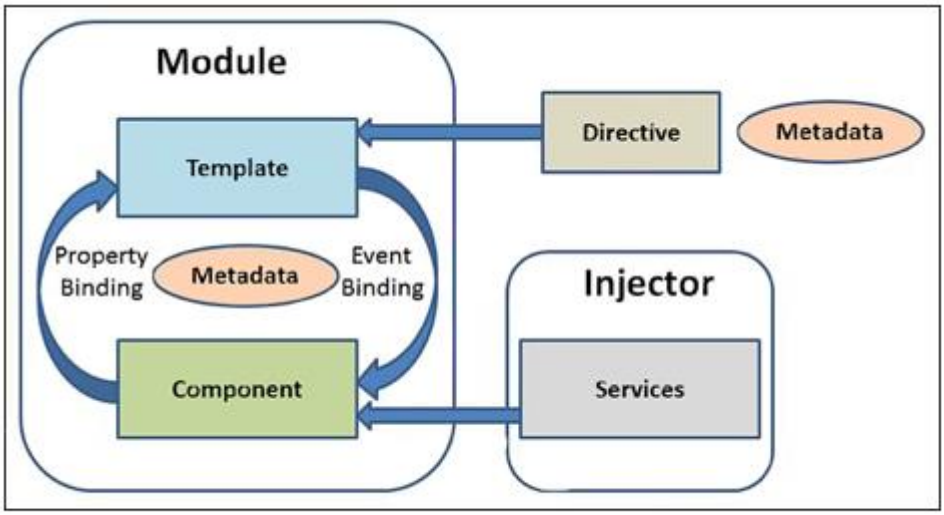

Рисунок 3.3 — Загальний огляд архітектури Angular

Варто зазначити, що Angular застосунки пишуться на TypeScript, а не на чистому JavaScript. Версія синтаксису для JavaScript не отримала широкого розповсюдження, тому на даний момент у документації усі синтаксичні конструкції описані з використанням синтаксису TypeScript.

Архітектура Angular складається з:

- Module;
- Component;
- Template;
- Service;
- Router;
- Pipe;
- Directives.

Модулі (Module) — структурні одиниці застосунку, які інкапсулюють певну логіку. В Angular це структури, які зберігають певні компоненти, директиви та сервіси, об’єднані певною логікою. Прикладом може слугувати профіль користувача, модуль для написання листа, огляд списку листів тощо.

Компоненти (Component) — typescript клас, який зберігає дані та логіку відображення цих даних у шаблоні (представленні). Шаблон тісно пов’язаний з компонентом. Дані з компонента можна з легкістю відображати у шаблоні, використовуючи спеціальний синтаксис. Компонент також може «знімати» дані з шаблона та отримувати їх безпосередньо у скрипті.

Шаблон (Template) — фрагмент html-коду з додаванням спеціального синтаксису. Він дозволяє впроваджувати в шаблон дані з компонента без використання innerHTML та подібних методів. Шаблон прописується у компоненті та є частиною його конфігурації.

Сервіс (Service) в Angular являє собою typescript класи, які виконують задачі, пов’язані з отриманням, зберіганням та обробкою даних. Наприклад, логування, перетворення даних для подальшої передачі у компонент, звернення до backend та ін. На відміну від компонентів та директив сервіси не працюють з представленнями (шаблонами) напряму. Задачі сервісів:

- Надання даних застосунку. Сервіс сам може зберігати дані у пам’яті або, з метою отримання даних, звертатися до якогось джерела даних, наприклад, до сервера;
- Сервіс може організувати канал взаємодії між окремими компонентами застосунку;
- Сервіс може інкапсулювати бізнес-логіку, різноманітні обчислювальні задачі, задачі з логування, які краще виносити поза компоненти. Таким чином, код компонентів буде зосереджений, безпосередньо, на роботі з представленням. До того ж, можемо розв’язати проблему повторення коду, якщо нам знадобиться виконати одну й ту саму задачу у різних

компонентах і класах.

Роутер (Router) — маршрутизатор, який призначений для переходу між екранами з метою відображення різного контенту. Іншими словами, коли в адресному рядку браузера змінюється фрагмент URL, маршрутизатор відстежує ці зміни та завантажує ту або іншу частину застосунку.

3.3 Express — Фреймворк для розробки серверної частини

Express являє собою популярний веб-фреймворк, написаний на JavaScript який працює всередині середовища виконання node.js. Цей модуль висвітлює деякі ключові переваги цього фреймворка, установку середовища розробки і виконання основних завдань веб-розробки і розгортання[5].

Express.js, або просто Express, фреймворк web-додатків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроектований для створення веб-додатків і API. Є стандартним каркасом для Node.js. Автор фреймворка описує його як створений на основі написаного на мові Ruby каркаса Sinatra, маючи на увазі, що він мінімалістичний і включає велику кількість додаткових плагінів. Express може бути backend'ом для програмного стека MEAN, разом з базою даних MongoDB і каркасом Vue.js, React або AngularJS для frontend'a.

3.4 Платформа Heroku для розгортання додатку

Heroku — хмарна платформа-сервіс (PaaS), яка підтримує декілька мов програмування. Є однією з перших хмарних платформ і розробляється з 2007 року. Було використано для розгортання веб додатку і сервера аутентифікації. Програми, що працюють на Heroku, використовують також DNS-сервер Heroku (зазвичай додатки мають доменне ім'я виду «ім’я_програми.herokuapp.com»). Для кожної програми виділяється кілька незалежних віртуальних процесів, які

називаються «dynos». Вони розподілені по спеціальній віртуальній сітці («dynos grid»), яка складається з декількох серверів (Рис. 3.4). Heroku також має систему контролю версій Git. Heroku це платформа-сервіс, яка базується на керованій контейнерній системі, з інтегрованими службами передачі даних та потужною екосистемою для розгортання та роботи сучасних додатків. Heroku — це сервіс, інтегрований з найпопулярнішими сьогоднішніми інструментами розробників та робочими процесами. Heroku запускає програми через dynos — надійні контейнери, які повністю керуються у середовищі виконання. Розробники розгортають свій код, написаний в Node, Ruby, Java, PHP, Python, Go, Scala або Clojure, в build-системі, яка і збирає додаток, готовий до виконання. Системні та мовні стеки відстежуються, виправляються та модернізуються, тому вони завжди готові та актуальні. Під час виконання програма працюватиме автоматично, без будь-яких ручних налаштувань.

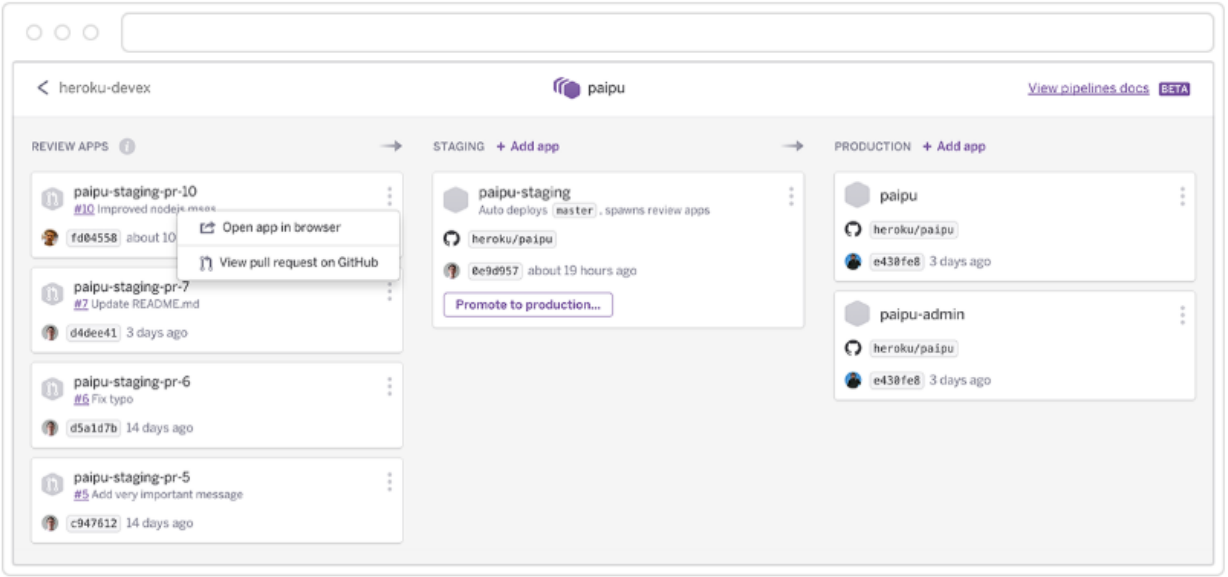

Рисунок 3.4 — Інтерфейс Heroku Pipelines

3.5 Система керування базами даних MongoDB

Документо-орієнтована система керування базами даних (СКБД) MongoDB є платформою з відкритим вихідним кодом, яка, на відміну від SQL,

не потребує опису схеми таблиць[6] для збереження даних. MongoDB займає нішу між системами, які зберігають дані в форматі ключ-значення та реляційними СКБД, які зручні для формування складних запитів.

Вибір даної бд обумовлений її тісним зв'язком з екосистемою NodeJS а також нативною підтримкою JSON – основного способу зберігання даних в мові JavaScript.

Документо-орієнтована система керування базами даних є системою керування базами даних яка спеціально призначена для зберігання ієрархічних структур даних, тобто документів, та найчастіше реалізована за допомогою підходу NoSQL. Основою документо-орієнтованих СКБД є документні сховища (англ. document store), які мають структуру дерева. Деревна структура починається з кореневого вузла і може містити декілька внутрішніх та відповідно листових вузлів.

Дані при додаванні в колекцію індексуються, це дозволяє навіть при досить складній структурі даних виконувати досить швидкий пошук. API для пошуку надає змогу здійснювати пошук документів або їх частин. На відміну від сховищ типу ключ-значення, вибірка за запитом до сховища документів може містити лише частини великої кількості документів без необхідності повного завантаження цих документів до оперативної пам'яті.

Декілька документів можна об'єднати(згрупувати) до колекції, колекціям документів в реляційних базах даних відповідає таблиця, але з деякими відмінностями, такими як те що документи в колекції можуть містити інші колекції в собі. Хоча структура документів в колекції може бути довільною, для більш ефективного індексування і пошуку рекомендується зберігати в колекціях документи, які мають спільну структуру. Документо-орієнтовані бази даних застосовуються у системах менеджменту контента, видавничій справі, для організації пошуку документів, тощо.

Центральною сутністю документо-орієнтованих баз даних є Документ. Визначення цієї головної концепції різниться в різних реалізаціях документо-орієнтованих баз даних, але загально всі вони стверджують те що документ

ізолює та зберігає дані у певному форматі або кодуванні. Це кодування може використовувати текстові формати зберігання даних такі як: XML, YAML, JSON, BSON, а також різноматніті бінарні формати зберігання інформації.

Документи всередині документо-орієнтованих баз даних певною мірою подібні до рядків або записів в реляційних базах даних, але без жорстких вимог до змісту даних. Документи не не мають необхідності дотримання стандартної схеми, ані мати однакові секції, частини або ключі. В реляційних базах даних, кожен запис має містити однакові поля, а невикористані поля залишаються пустими або заповняються стандартими значеннями, на противагу цьому в документно-орієнтованих базах даних немає пустих полів в жодному документі (записі). Такий підхід дозволяє змінювати певний документ без необхідності решти документів колекції мати таку ж саму структуру.

Документо-орієнтована СКБД для ідентифікації документа використовує унікальний ідентифікатор, який представляє документ. В якості ключа часто використовуєтсья рядковий ідентифікатор. Для прискорення виконання запитів до колекції СКБД зберігає індекс унікальних ідентифікаторів.

Ще однією характеристикою документо-орієнтованих систем керування базами даних є спосіб пошуку документів за схемою ключ-документ. Різні документо-орієнтовні мають значно відмінний на противагу реляційним базам даних програмний інтерфейс (API) або мову запитів, котрі надають користувачеві засоби шукати документ. Наприклад, необхідно запитати всі документи з певним набором полів і з певними значеннями. Набір інтерфейів для запитів та мов для формування запитів у різних реалізаціях, так само як продуктивність запитів, може значно відрізнятися в залежності від реалізації документо-орієнтованої СКДБ.

Різні імплементації документо-орієнтовних СКБД відрізняються способами організації документів, включаючи такі розповсюджені як:

- Зберігання даних в колекціях;
- Використання тегів для документів;
- Використання невидимі метадані;

- Побудова ієрархії директорій;
- Комірки.

MongoDB — дуже популярна Open Source система керування базами даних, в якій дані зберігаються як колекції (групи) документів. Документи представлені в зрозумілому формат JSON. В MongoDB різні документи однієї і тієї ж колекції можуть мати різні поля і типи — немає однієї схеми для всіх.

3.6 Середовище розробки Visual Studio Code

У якості середовища розробки було обрано Visual Studio Code — засіб для створення, редагування сучасних веб-додатків і застосунків для хмарних систем. Visual Studio Code є безкоштовним програмним забезпеченням і доступний у версіях для Windows, Linux і OS X. Компанія Microsoft представила Visual Studio Code у квітні 2015. Це середовище розробки стало першим кросплатформним продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проекту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js (Рисунок 3.5).

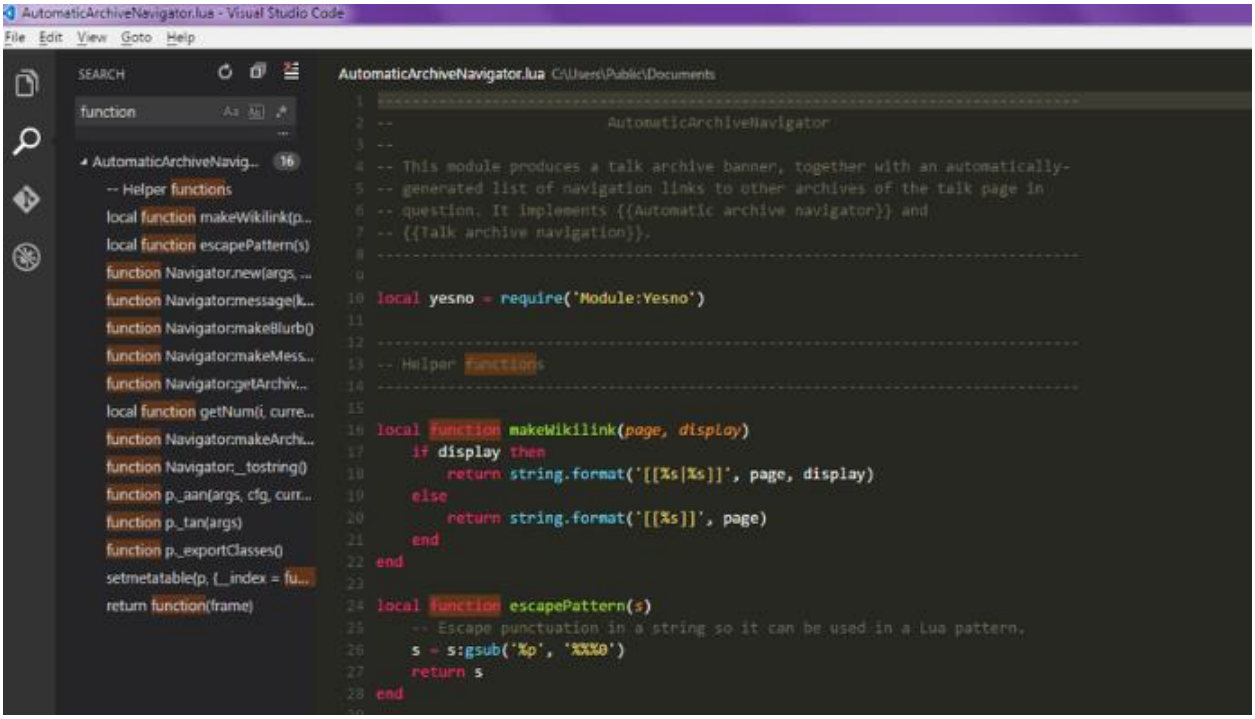

Рисунок 3.5 — Приклад програми розробленої у Visual Studio Code

Visual Studio Code підтримує ряд мов програмування, підсвічування синтаксису, IntelliSense, рефакторинг, налагодження, навігацію по коду, підтримку Git та інші можливості. Багато можливостей Visual Studio Code недоступні через графічний інтерфейс, найчастіше вони використовуються через палітру команд або JSON файли (наприклад, призначені для користувача налаштування). Палітра команд представляє собою командний рядок, який викликається поєднанням клавіш. Visual Studio також дозволяє змінювати кодову сторінку при збереженні документа, символи перекладу рядка і мову програмування поточного документа. Існує також розширення Python для Visual Studio Code з відкритим вихідним кодом. Воно надає розробникам широкі можливості для редагування, налагодження і тестування коду. На даний час за допомогою вбудованого в продукт призначеного для користувача інтерфейсу можна завантажити і встановити кілька тисяч розширень тільки в категорії «programming languages» (мови програмування).

Visual Studio Code збирає дані про використання і відправляє їх в

Microsoft, але ця функція не є обов'язковою (опція File> Preferences> Settings, «telemetry.enableTelemetry»). Хоча надання даних можна відключити відмовитися від передачі персональних даних, деякі можливості, такі як персоналізація, що використовують такі дані, будуть недоступні для відключення. Дані можуть передаватися контрольованим філіям Microsoft, дочірнім компаніям і правоохоронним органам відповідно до заяви про конфіденційність. Visual Studio Code є простим, але потужним редактором початкового коду. За первинною конфігурацією використовується для редагування коду на JavaScript, TypeScript і Node.JS, а за допомогою розширень підтримує C++, C#, Python і PHP. Visual Studio Code також виконує автодоповнення: за допомогою технології IntelliSense дописує назви оголошених змінних, функцій і модулів, а також робить посилання на відповідний розділ документації. Можлива відладка коду безпосередньо з редактора, запуск додатка для відладки і приєднання до запущених додатків.

3.7 Висновки до розділу

Для розроблення системи для двофакторної авторизації на основі TOTP алгоритму з використанням d-HMAC хеш функції для отримання одноразових паролів було обрано мову програмування JavaScript, фреймворк Angular, платформу з відкритим кодом NodeJS та систему керування базою даних MongoDB. Програмні компоненти розроблені з використанням даних програмних засобів буде розгорнуто в хмарному середовищі Heroku. Все використане програмне забезпечення має необхідні інструменти для реалізації системи двофакторної авторизації.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розроблений програмний засіб складається з трьох основних частин:

- Сервер аутентифікації;

- Клієнтський додаток;
- Додаток для генерування одноразових паролів.

Користувач при реєстрації в системі отримує до неї частковий доступ, для того щоб отримати повний доступ до системи він надсилає запит для реєстрації двохфакторної авторизації. При цьому система відсилає запит на сервер автентифікації для генерації секретного ключа, далі згенерований секретний ключ відправляється до системи і відображається для користувача у вигляді QR-коду. Користувач за допомогою мобільного додатку зчитує QR-код. При повторному вході в систему користувач має змогу крім паролю ввести ще й токен, який генерується в мобільному додатку на основі часової мітки і секретного ключа. При введені паролю і токена автентифікації система відправляє токен користувача на сервер автентифікації, сервер автентифікації використовуючи той самий секретний ключ генерує токен, якщо токен користувача і токен згенерований на сервері збігаються то сервер автентифікації надсилає до системи результат автентифікації. При успішному вході користувач отримує повний доступ до системи.

4.1 Опис покращеного HMAC алгоритму

Основними цілями проектування функцій HMAC є такі:

• HMAC може використовувати будь-який тип хеш-функції. Більшість хеш-функцій доступні безкоштовно.

• HMAC не впливає на хеш-функцію, тому і не має негативного впливу на її продуктивність.

• HMAC надає перевагу легко доступним ключам.

Алгоритм HMAC простий та його можна легко змінювати, дозволяючи на вимогу підвищити рівень безпеки або швидкість основної хеш-функції.

Припустимо, що HMAC може використовувати будь-яку хеш-функцію h без її модифікацій та секретний ключ K. Функція Hash h буде виконуватися на основі функції стиснення ϕ для повідомлення m, що містить l блоків. Довжина

кожного блоку l у бітах позначається b, що вказує, що l * b буде дорівнює довжині повідомлення m побітового вирівнювання. Позначимо через n довжину дайджесту повідомлення в бітах. У своєму дослідженні ми використовуємо функції SHA-256, в результаті чого n = 256. Довжина загального секретного ключа K може дорівнювати b або менше; для ключів, довших за b біт, слід обчислити хешування за допомогою функції h. Дизайнери HMAC рекомендують використовувати K, що має принаймні n біт. Використовувати ключі довжиною менше n не рекомендується, оскільки це зменшує криптографічну стійкість HMAC.

Розробники HMAC рекомендують використовувати високо випадкові ключі, і часто змінювати ключі як стандартну практику безпеки. Це мінімізує негативні наслідки відкритих ключів та зменшує загрози, які залежать від збору даних, розрахованих HMAC, з використанням того самого ключа, а також атаки грубою силою.

Досить ефективним є покращення хеш-функцій для можливості використання динамічних вхідних векторів замість статичних значень. Крім того, HMAC обчислює проміжні значення (K+ $\oplus$ opad) and (K+ $\oplus$ ipad) ще один раз для того ж K. Ці проміжні значення будуть використовуватися багато разів для автентифікації одного і того ж ключа; в результаті ці значення, включаючи секретні ключі, повинні бути захищені від будь-якого виду розкриття.

Покращений d-HMAC оснований на HMAC і працює в схожій манері. Його відмінності мають мету покращити стійкість HMAC від різних видів атак. Одна з його переваг це те, що немає необхідності зберігання проміжкових значень, тому що ці значення динамічно розраховуються для різних повідомлень. І тому немає необхідності в реалізації техніки захисту їх від неавторизованого доступу.

Одна з відомих атак на HMAC полягає в тому щоб зібрати якнайбільше хеш результатів згенерованих одним й тим самим секретним ключем, це дозволяє зловмиснику розпочати офлайн атаку з підбора секретного ключа. В противагу з

d-HMAC не так просто зібрати різні повідомлення згенеровані одним і тим самим секретним ключем, тому що, ipad та opad завжди генеруються динамічно. Як результат d-HMAC більш криптографічно стійкий ніж HMAC. Більше того HMAC функція має надійно зберігати проміжні значення, коли покращений d-HMAC не має ніяких криптографічних правил або засобів, щоб зберігати проміжні значення (K+ ⊕ ipad') та (K+ ⊕ opad'), так як ці значення розраховуються динамічно. Програмісти, які використовують HMAC для автентифікації мають забезпечити безпечний метод зберігання проміжних значень, що може призвести до серйозних вразливостей. В покращеному d-HMAC, немає необхідності для цих мір.

d-HMAC також не втрачає корисних криптографічних характеристик, результати тестів показують, що він не втрачає лавинного ефекту або балансувальних властивостей.

Ефект лавини – це одна з найголовніших цілей в проектуванні хеш функцій і будь-яких криптографічних засобів. Якщо вхідне повідомлення, або секретний ключ має незначні зміни приблизно половина вихідних бітів буде відрізнятися. Приблизно 10000 випадкових тестових повідомлень було протестовано для HMAC та d-HMAC. Кожен результат тесту в таблиці 2.2 відображає ефект від зміни одного біту повідомлення, додатково до цього було протестовано 10000 випадкових ключів, в кожному тесті було змінено один біт в ключі, результати зображено в таблиці 4.1. Таблиця 4.1 Лавинний ефект після зміни одного біта повідомлення

XOR результати	HMAC (bits)	d-HMAC (bits)
Мін.	97	97
Макс.	157	159
Середній	128.003	127.956

Таблиця 4.2 Лавинний ефект після зміни одного біта ключа

XOR результати	HMAC (bits)	d-HMAC (bits)
Мін	103	105
Макс	152	150
Середній	127.775	128.295

Ефект балансування це інша важлива якість хеш функцій і криптографічних засобів в якому вихідна комбінація має мати приблизно однакову кількість символів зі словника (0 та 1). Десять тисяч випадкових вхідних данних було протестовано з HMAC та d-HMAC. В цих тестах, один біт вхідного повідомлення (таблиця 2.4) було змінено кожного разу так само як і біт секретного ключа (таблиця 2.5). Результати тестів не показують значної різниці між двома алгоритмами.

Таблиця 4.3 Результати балансування при зміні біта вхідного повідомлення

XOR results	HMAC (bits)	d-HMAC (bits)
Мін.	98	98
Макс.	162	157
Середній	127.962	128.062

Таблиця 4.4 Результати балансування при зміні біта ключа

XOR результат	HMAC (bits)	d-HMAC (bits)
Мін.	106	106
Макс.	149	149
Середній	127.829	127.899

З результатів тестів, можна зробити висновок, що модифікації зберегли корисні криптографічні властивості оригінального алгоритму, а в деяких випадках його їх покращили, єдиний мінус полягає в тому що модифікований алгоритм трішки повільніший через необхідність додаткових обчислень.

Вводимо дані умовні позначки:

1. T – Дискретне значення часу, яке використовується в якості параметру.
2. X – інтервал часу протягом якого дійсний пароль (30 с.)
3. T0 – початковий час, для синхронізації сторін.
4. K – спільний секрет.
5. CurrentTime – поточний час.

T = (CurrentTime – T0)/X

HOPT(K, T) = Truncate(d-HMAC-SHA-1(K,T))

TOTP = HOPT(K, T)

Удосконалений d-HMAC працює аналогічно HMAC, але використовує динамічні значення для ipad та opad замість стандартних фіксованих значень. Розрахунки для ipad та opad залежать від трьох параметрів: таблиці підстановок S-box вмісту повідомлення (в даному випадку часової мітки), та спільного ключа. Очікується, що покращений d-HMAC буде більше стійким до криптографічних атак, оскільки він використовує динамічні значення ipad та opad.

Алгоритм генерації ipad та opad

1. $w = h(m)$
2. $ip = w \oplus e_R$
3. A="", B="", g=0.
4. for i= 1 to n/4 do begin $g=g+1$ x=""; y="";
   for j=0 to 3 do begin $x = x \,||\, ip[(g*i)+j]$

   end;

   $y = x$

   s = T[Decimal(x)]; z = Binary(s), where length of z equals 4 bits $A = A \,||$ z v = T[Decimal(y)]; w= Binary(v), where length of w equals 4

   bits

   $B = B \,||\, w$ end,

5. *ipad' = A, opad' = B.*

1. Обчисліть дайджест повідомлення h (m) за допомогою хеш-функції SHA-256 на m.

2. Використати операцію XOR дайджеста повідомлення h (m) з eR.

3. Використати таблицю S-Box для обчислення ipad' та opad'.

Таблиця 4.6 (a) Результати балансування при зміні біта ключа

3	**2**	**1**	**0**	
12	9	3	5	**0**
9	3	10	9	**1**
6	5	12	6	**2**
10	12	6	3	**3**

Таблиця 4.6 (b) Результати балансування при зміні біта ключа

5	**0**
3	**1**
9	**2**
12	**3**
9	**4**
10	**5**
3	**6**
9	**7**
6	**8**
12	**9**
5	**10**
6	**11**
3	**12**
6	**13**
12	**14**
10	**15**

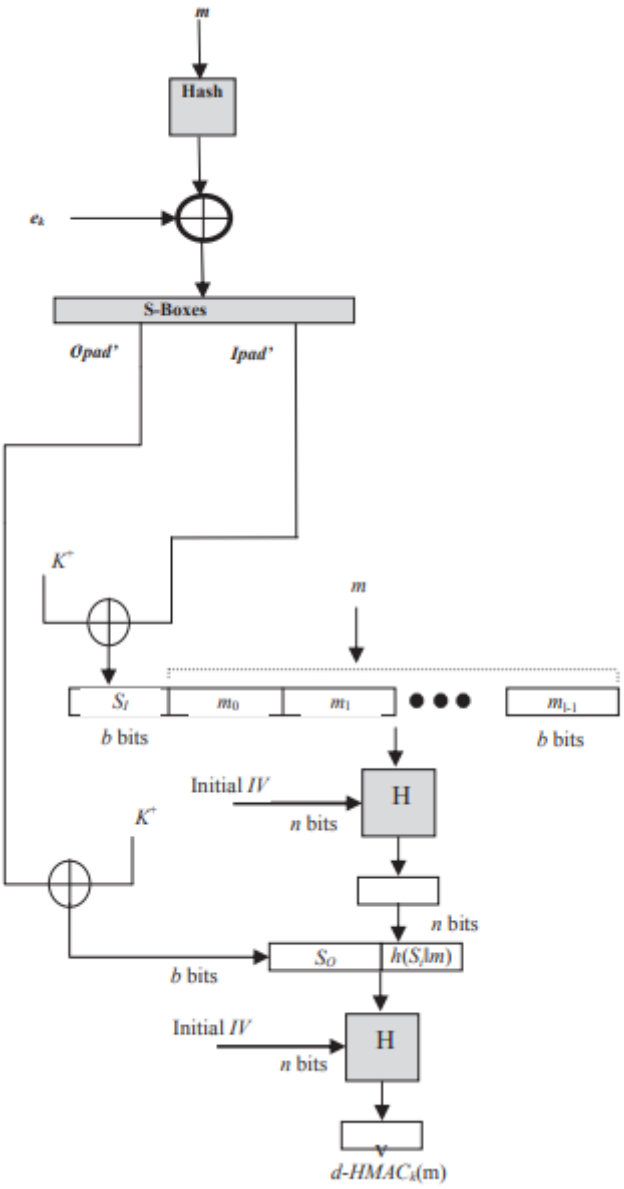

Рисунок 4.1 — Алгоритм обчислення d-HMAC

4.2 База даних

База даних MongoDB, яку було використано у даному програмному засобі, відноситься до документо-орієнтованих баз даних загального призначення. З її допомогою була спроектована база даних для системи сервера аутентифікації і для клієнтського додатку.

Для зручності роботи з документами БД було використано Mongoose, яка являє собою спеціальну ODM-бібліотеку (Object Data Modelling) для роботи

з MongoDB, яка надає змогу зіставляти об’єкти класів і документи з колекцій бази даних.

```
var userSchema = new mongoose.Schema({
  email: {
    type: String,
    unique: true,
    required: true
  },
  name: {
    type: String,
    required: true
  },
  hash: String,
  salt: String,
  tokenProvided: {
    type: Boolean,
    default: false
  },
  tokenValid: {
    type: Boolean,
    default: false
  }
});
```

Рисунок 4.2 — Схема документу користувача

Таблиця user (Рисунок 4.2) зберігає інформацію про користувача системи. Поля:

- email – електронна пошта користувача і його ідентифікатор;
- name – ім’я користувача;
- hash – хеш пароля користувача (зберігання паролю в зашифрованому вигляді - це погана практика, тому пароль зберігається у вигляді хешу);
- salt – сіль (також модифікатор) — рядок даних, який передається хеш-функції разом з паролем.

Головним чином використовується для захисту від перебору за словником і атак з використанням райдужних таблиць, а також приховування однакових паролів.

- tokenProvided – логічне значення, чи було користувачу надано секретний ключ для генерації одноразового паролю;
- tokenValid – логічне значення, чи було використано користувачем одноразовий пароль.

```
var tokenSchema = new mongoose.Schema({
  secretKey: String,
  userId: String,
  valid: Boolean,
  applicationId: String
});
```

Рисунок 4.3 — Схема колекції токенів аутентифікації

Таблиця token (Рисунок 4.3) використовується сервером аутентифікації та зберігає інформацію про видані секретні ключі системи. Поля:

- secretKey – секретний ключ для генерації одноразового паролю;
- userId – ідентифікатор користувача;
- valid – логічне значення, чи користувач отримав секретний ключ;
- applicationId – ідентифікатор додатку, для якого було згенеровано секретний ключ.

4.3 Сервер аутентифікації

Перевірку правильності одноразового пароля на стороні сервера

використовує спеціальний сервер аутентифікації, який програмно обраховує поточне значення одноразового пароля, для збереження принципу багатофакторної автентифікації користувач має крім пароля згенерованого додатков ввести і постійний пароль.

Сервер являє собою API додаток за допомогою якого кінцева система буде видавати секретні ключі і перевіряти достовірність одноразових паролів.

Список API, які надає система:

- post server_url/getToken - post запит на генерацію секретного ключа для користувача. В тілі запиту кінцева система надає свій власний ідентифікатор (користування системою може бути надано під платною підпискою) та власне ідентифікатор користувача в відповідь сервер генерує випадковий секретний ключ, який буде використовуватись при генерації одноразового паролю.
- post server_url/verifyToken - post запит на підтвердження користувачем отримання секретного ключа. Використовується для обліку згенерованих ключів і для оповіщення кінцевої системи про успішну реєстрацію.
- post server_url/submitToken - post запит для верифікації введеного користувачем одноразового паролю - результат цього запиту і буде використовуватись як підтвердження достовірності користувача.

4.4 Клієнтський додаток

Для демонстрації роботи системи було розроблено базовий клієнтський додаток, який буде інтегруватися з сервером аутентифікації. Додаток являє собою веб застосунок, містить логіку реєстрації користувачів, здійснює надсилання одноразових паролів на сервер аутентифікації і на основі результатів цих запитів робить відповідні дії. Тип клієнтського додатку не має значення, сервер аутентифікації можна використовувати в консольних, дестктопних,

браузерних додатках. Список API, які реалізовано в демо додатку:

- get application_url/profile - get запит на отримання даних профілю користувача.
- get application_url/authToken - get запит на генерацію секретного ключа для генерації одноразових паролів, результат запиту відображається для користувача у вигляді QR коду, для читання додатком генерації ключів.
- post application_url/verifyToken - post запит для перевірки того, що користувач отримав секретний ключ.
- get application_url/protected - get запит який слугує прикладом закритого ресурсу, для доступу до якого користувачу необхідно здійснити вхід до системи з використанням одноразового паролю. У випадку відсутності ключа, або якщо він неправильний користувач не буде мати доступу до даного ресурсу.
- post application_url/register - post запит, для реєстрації нових користувачів.
- post application_url/login - post запит для входу в систему, користувач може увійти до системи без одноразового паролю лише з обмеженим доступом.

4.5 Додаток для генерації одноразових паролів

Для генерації одноразових паролів і збереження секретного ключа на стороні користувача було розроблено додаток, який має змогу за допомогою камери телефону зчитувати секретний ключ, а потім за вимоги за допомогою того ж алгоритму, що і сервер аутентифікації обраховує поточне значення одноразового паролю і відображає його користувачу для подальшого використання.

Для правильного функціонування додатку необхідно з певною точністю синхронізувати годинники сервера аутентифікації та додатку для генерації

паролів, на щастя більшість сучасних телефонів постійно синхронізують годинники за допомогою мережі інтернет, для правильного функціонування SSL також необхідно щоб на користувацькому девайсу був правильно заданий час.

Ще одним способом уникнення необхідності жорсткої синхронізації часу є реалізація на сервери аутентифікації перевірки ключа не в певний конкретний період часу, а з певним допуском, наприклад, до трьох відрізків часу (1.5 хв).

4.6 Висновки до розділу 4

Було реалізовано покращений HMAC алгоритм, перевірено його працездатність в мобільному веб додатку, а також на сервері автентифікації. Компоненти системи було поєднано між собою за допомого REST API і розгорнуто в хмарному середовищі. Було реалізовано і розгорнуто три додатки, а також дві бази даних для функціонування додатків.

Як базу даних було використано MongoDB. Його плюси це – безкоштовність, швидкість роботи, розповсюдженість, легка інтеграція з мовою програмування JavaScript а також можливість її швидкого розгортання засобами Heroku. Компоненти було скомпоновано один з одним а також перевірено їх працездатність.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Користувач під час користування системою явно взаємодіє лише з кінцевою системою а також додатком для генерації паролів. Посередником для спілкування з кінцевою системою виступає сам клієнтський додаток. Додаток для генерації одноразових паролів являє собою мобільний застосунок, який надає змогу зберігати секретний ключ згенерований сервером аутентифікації, а також генерувати одноразові паролі.

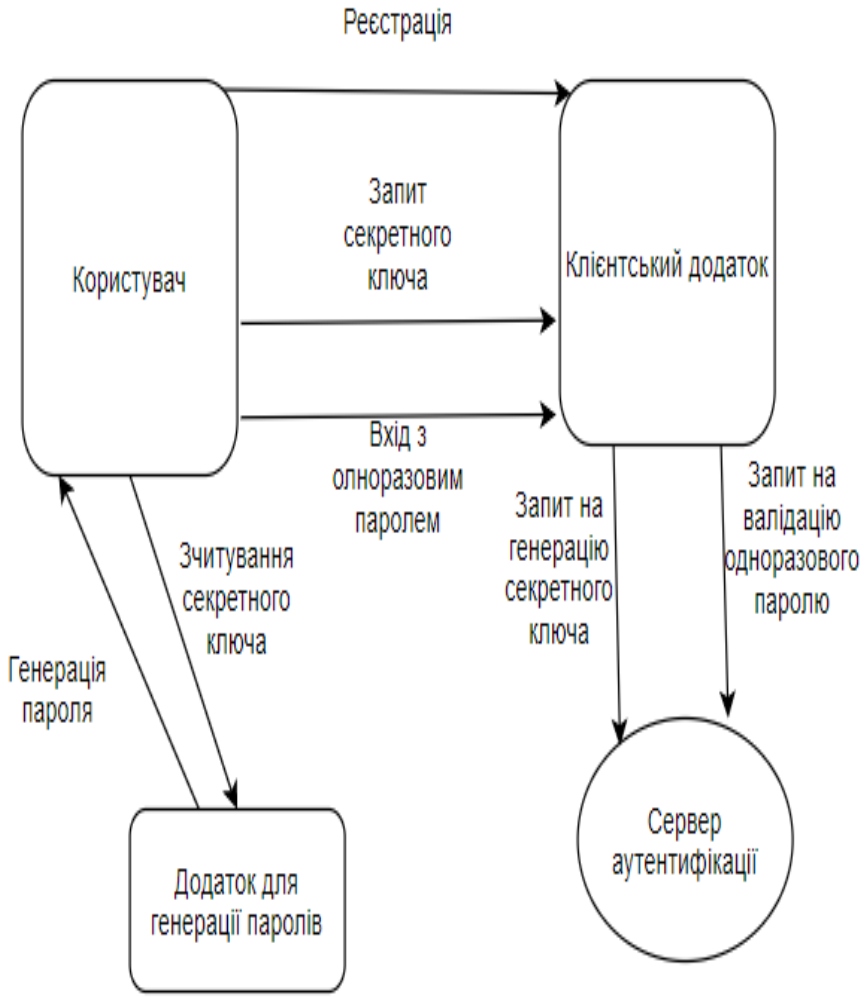

Рисунок 5.1 — Схема роботи системи

5.1 Реєстрація в системі

Для демонстрації роботи системи було розроблено додаток в якому можна зареєструвати користувачів. Під час реєстрації користувач має ввести ім’я користувача, свою електронну пошту, а також пароль в відповідні поля користувацького інтерфейсу (Рисунок 5.2.).

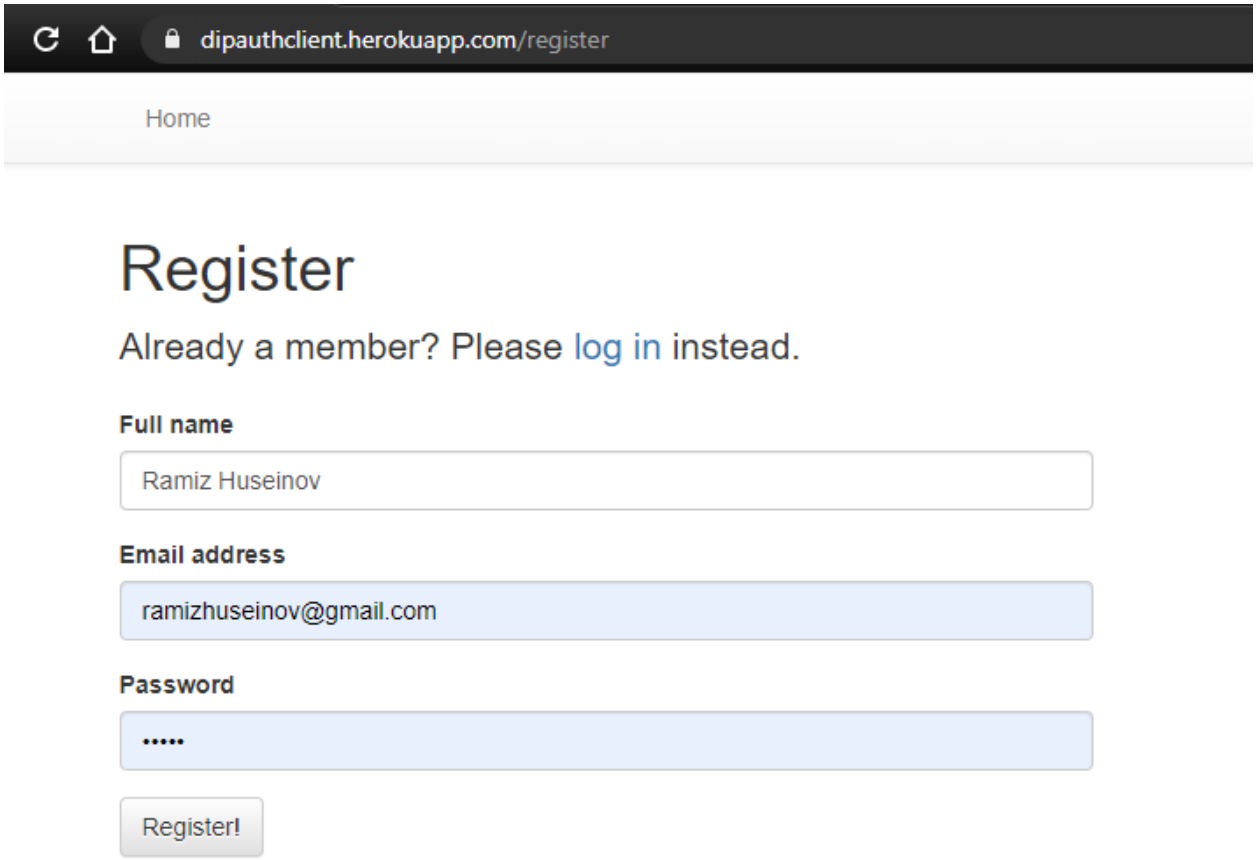

Рисунок 5.2 — Форма реєстрації користувача

При натисненні кнопки “Register” дані форми надсилаються на сервер та зберігаються до відповідної колекції бази даних MongoDB.

Використавши дані, які користувач надає при реєстрації до системи він може здійснити до неї вхід за допомогою відповідної форми, ця форма містить додаткове поле “Token” (Рисунок 5.3), в яке користувач має ввести одноразовий пароль згенерований в додатку, для отримання обмеженого доступу до системи

дане поле можна залишити пустим.

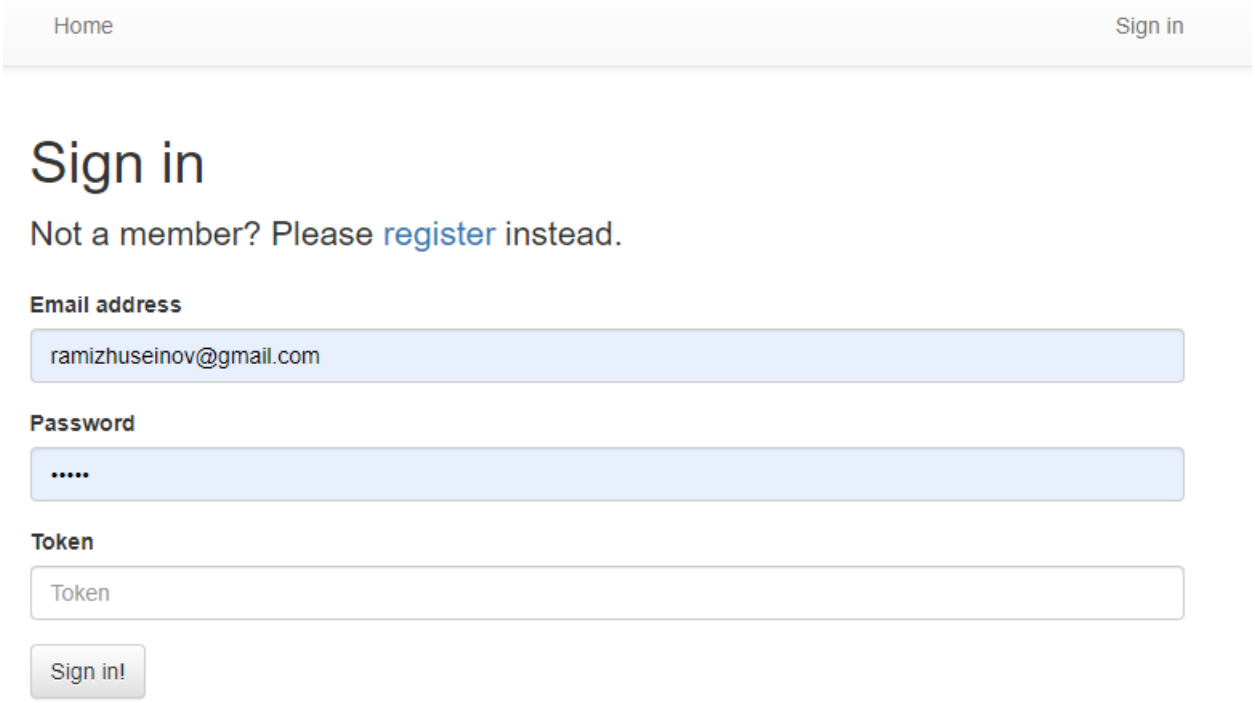

Рисунок 5.3 — Форма входу користувача до системи

5.2 Реєстрація секретного ключа в додатку

Для читання, збереження секретного ключа було розроблено окремий додаток, який надає змогу генерувати одноразовий пароль. Даний додаток є чисто клієнтським, тобто йому немає необхідності доступу до мережі інтернет.

Додаток було розроблено з використанням фреймворку Angular і розгорнуто як веб сервіс на основі Heroku. Інтерфейс розділений на 2 частини: вікно зчитування секретного ключа та вікно з одноразовим паролем (Рисунок 5.1).

Рисунок 5.4 — Вікно зчитування секретного ключа

Для роботи додатку користувач має надати згоду на використання камери додатком. Користувач після реєстрації в системі робить запит на генерацію секретного ключа і після цього клієнтський додаток відображає його в вигляді QR-коду.

При успішному зчитуванні секретного ключа, відповідний статус відображається на екрані.

Рисунок 5.5 — Вікно генерації одноразового паролю

Після зчитування ключа, користувач має змогу відкрити вікно, яке відображає одноразовий пароль. А також приблизний час, протягом якого даний ключ буде дійсним. Для того щоб отримати повний доступ до системи користувач після реєстрації має увійти до системи, створити запит на секретний ключ і використовуючи відповідну форму підтвердити факт отримання ключа.

5.3 Висновки до розділу 5

Було переглянуто загальні функції демо додатку а також системи двохфакторної автентифікації, а також їхню взаємодію з користувачем а також розглянуто загальну схему взаємодії всієї системи в цілому.

6. ВИПРОБУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАСОБУ

Для випробування додатку було зареєстровано нового користувача системи (Рисунок 6.1).

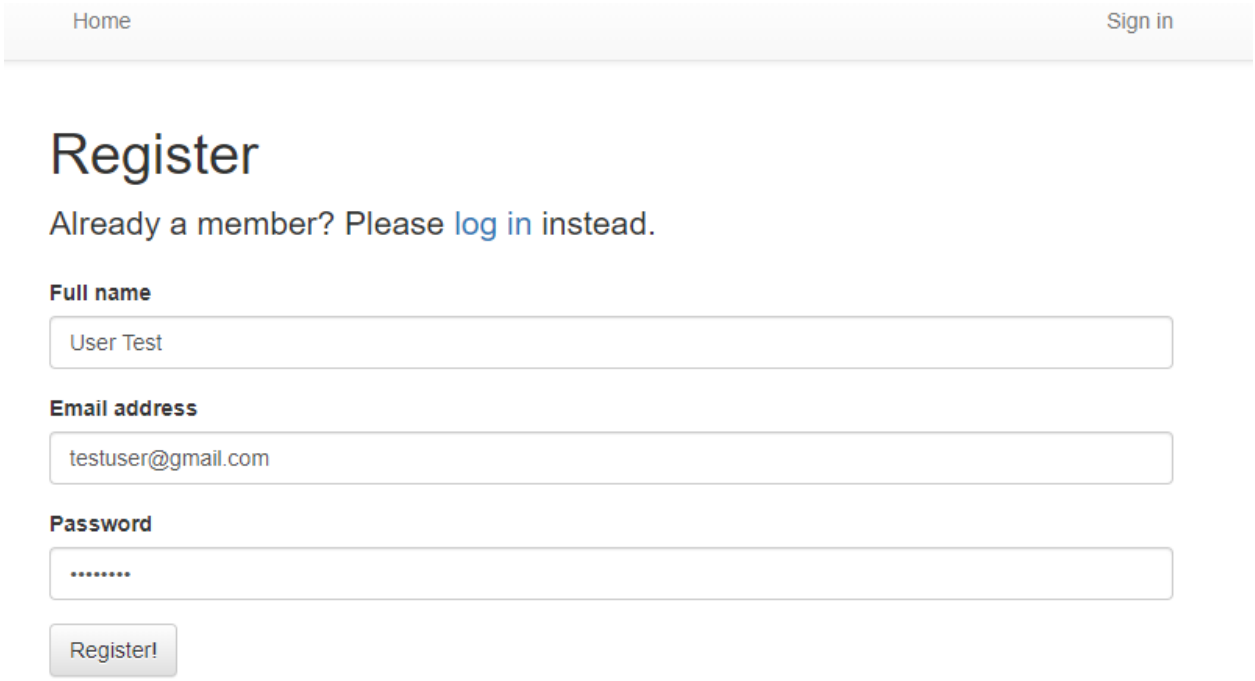

Рисунок 6.1 — Реєстрація користувача

Після реєстрації і входу до системи користувач має змогу зробити запит на генерацію секретного ключа вибравши відповідний пункти меню.

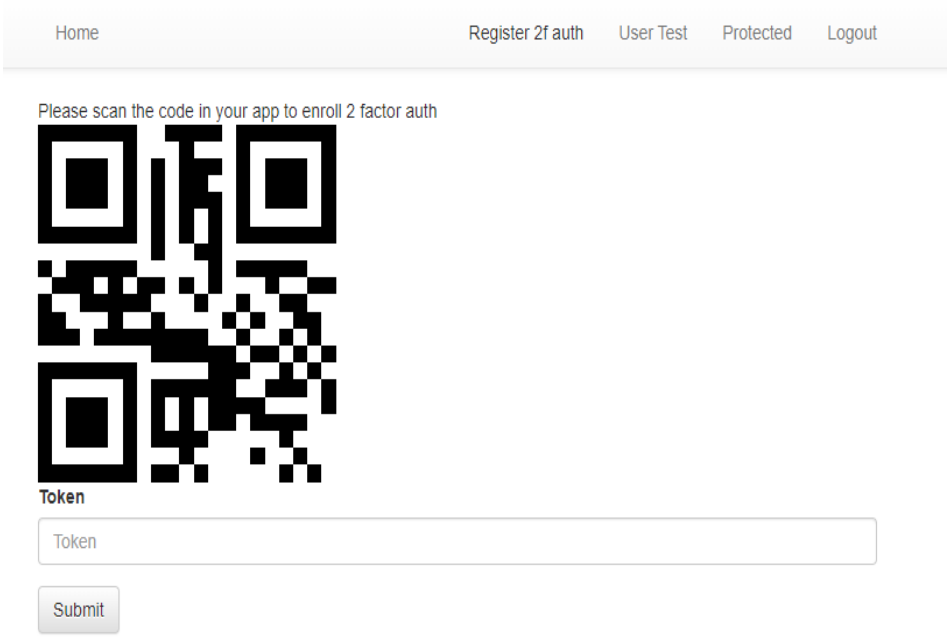

Рисунок 6.2 — Генерація ключа

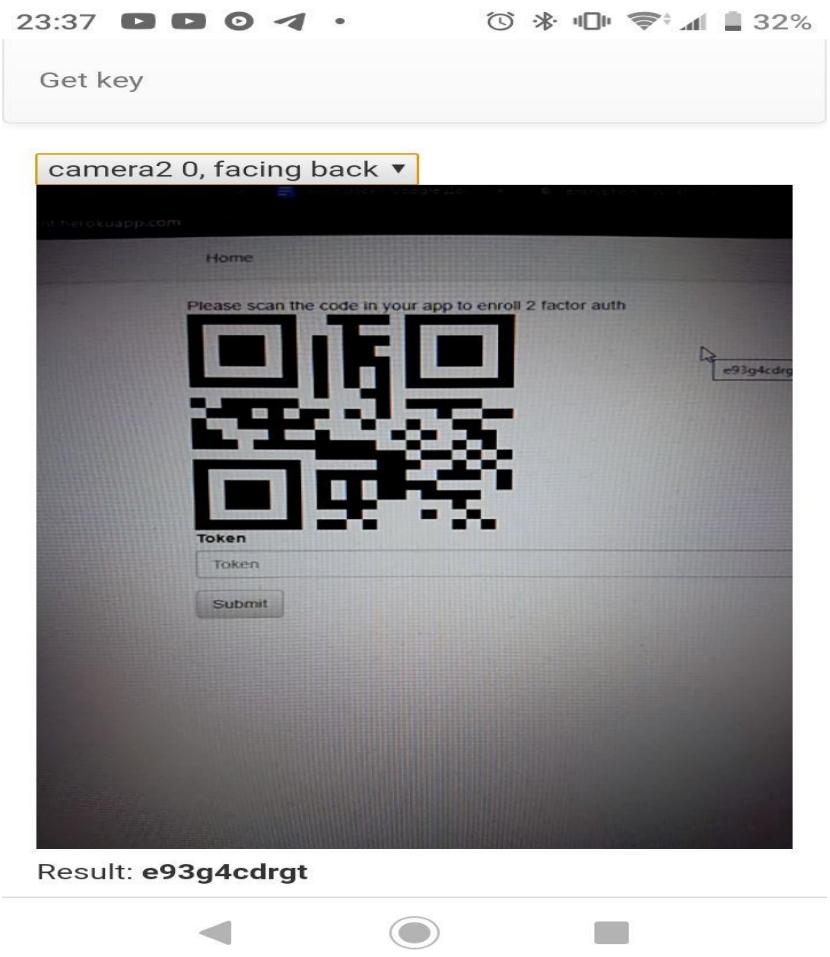

Рисунок 6.3 — Зчитування ключа

Зчитавши згенерований ключ користувач може перейти до меню

генерації одноразового пароля і здійснити вхід до системи, надіславши окрім постійного пароля ще й одноразовий. Якщо одноразовий пароль згенерований користувачем збігається з паролем, який буде згенеровано на сервері автентифікації то користувач отримує повний доступ до системи.

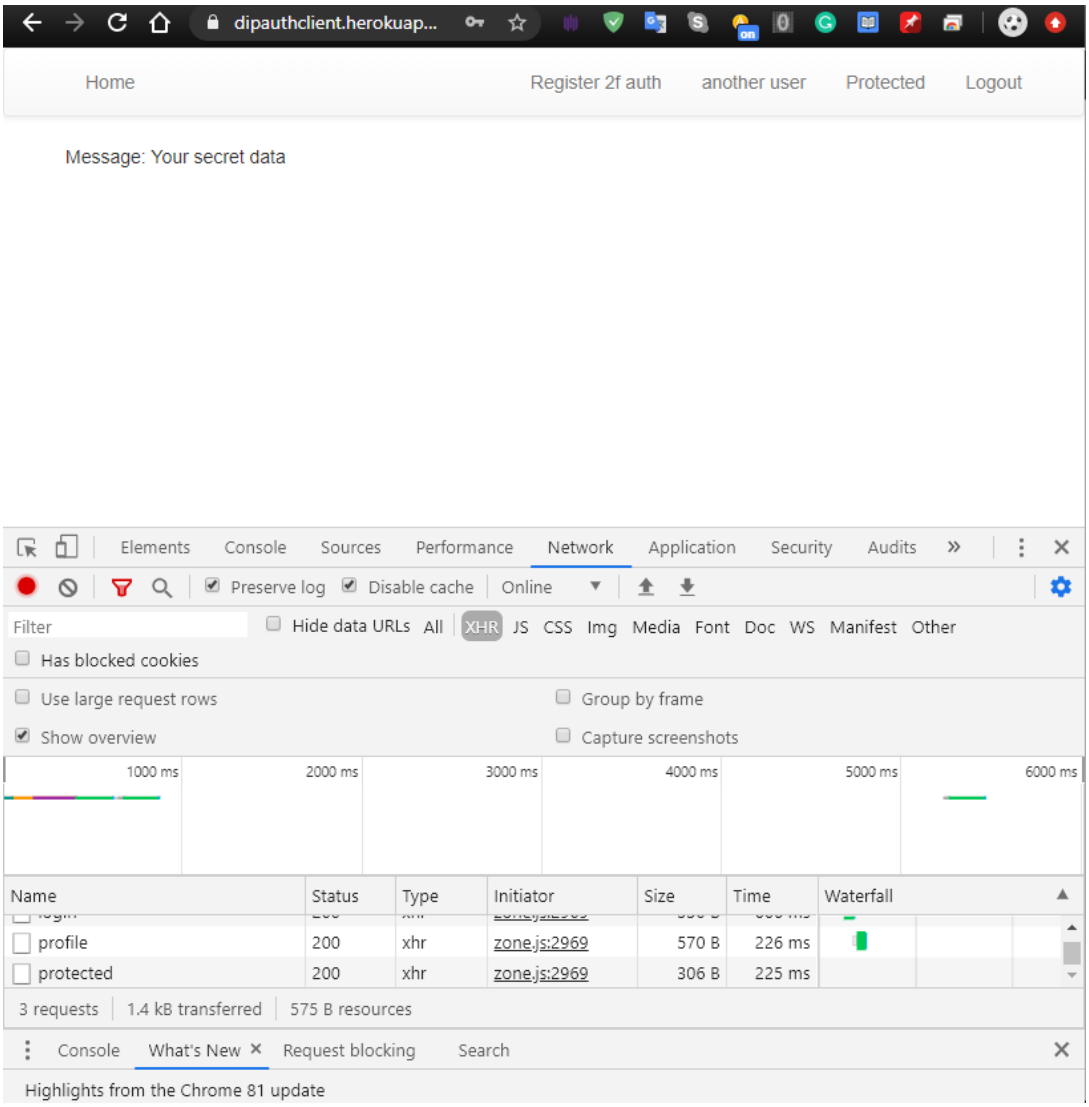

Рисунок 6.3 — Доступ до захищених даних

У випадку, якщо користувач надав невірний одноразовий пароль система відхилить його запит (Рисунок 6.4).

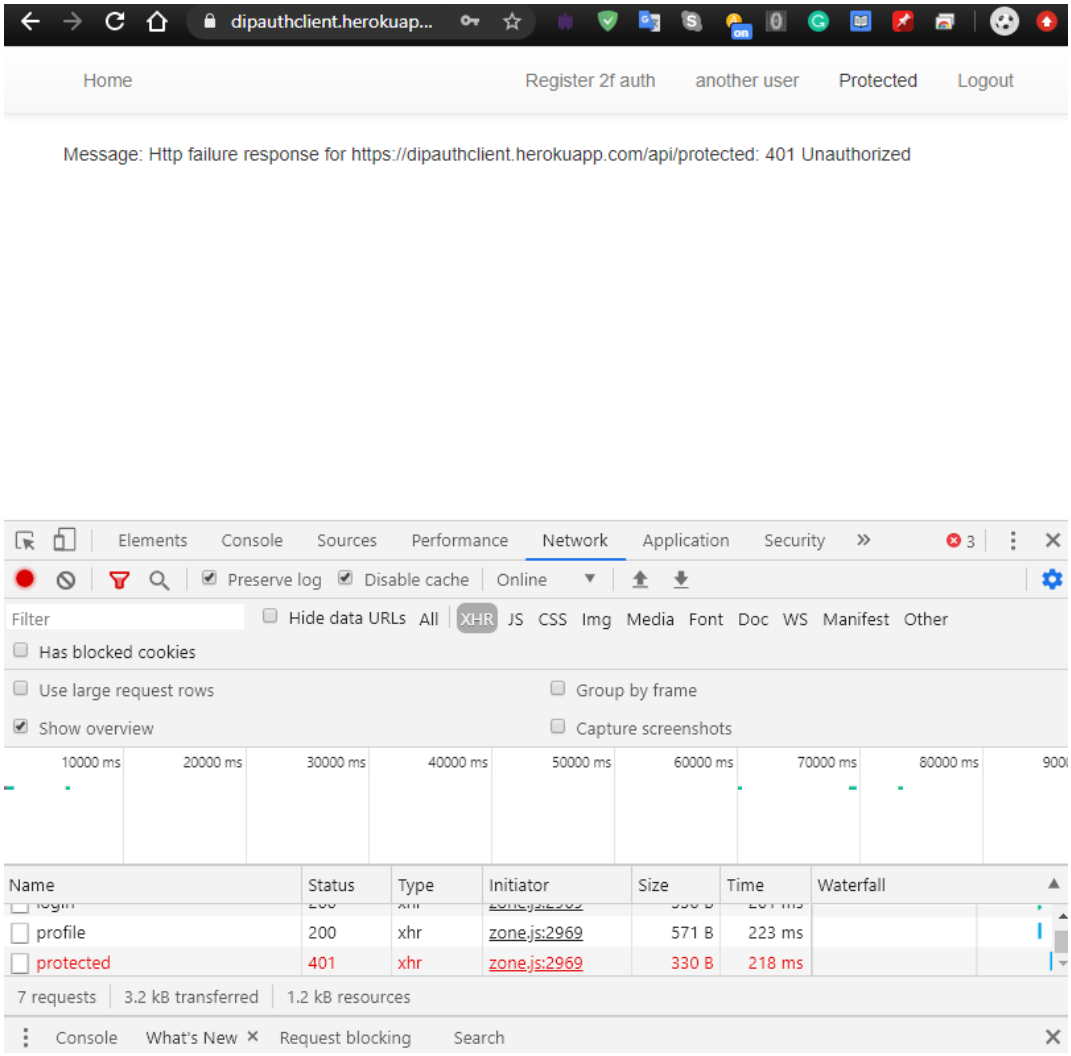

Рисунок 6.4 — Відхилення запиту

6.1 Висновки до розділу 6

У ході випробування системи аутентифікації було перевірено всі етапи роботи системи - реєстрація користувача, реєстрація секретного ключа а також генерація і використання одноразового паролю. Система була створена та перевірена на працездатність.

ВИСНОВКИ

В процесі виконання роботи було розглянуто підходи до підвищення захисту програмних продуктів, з використанням сучасних програмно-технічних рішень, методи розгортання і взаємодії декількох незалежних додатків, навики розробки з використанням JavaScript, Angular, NodeJS, Express та Heroku.

Було реалізовано покращену HMAC функцію а також використано її як основу системи двохфакторної автентифікації. Була спроектована, запрограмована і розгорнута система для двофакторної авторизації на основі TOTP алгоритму з використанням покращеної HMAC функції для отримання одноразових паролів.

Отриманий додаток може застосовуватися на реальних проектах для забезпечення більш безпечної аутентифікації користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Najjar, M. and Najjar, F., “d-HMAC Dynamic HMAC Function. Dependability of Computer Systems”, DepCos-RELCOMEX '06. International Conference on, 119-126, DOI: 10.1109/DEPCOSRELCOMEX, 2006

2. Krawczyk, H., Bellare, M., Canetti, R., “HMAC: Keyed-Hashing for Message Authentication”, RFC 2104, 1997.

3. Optimalidm [Електронний ресурс] – Режим доступу до ресурсу: https://optimalidm.com/resources/blog/guide-to-authentication/

4. Integrity, internal control and security in information systems: Connecting governance and technology ed. Michael Gertz, Erik Guldentops, Leon Strous ISBN 1-4020-7005-5

5. Rugged Embedded Systems Computing in Harsh Environments Augusto Vega Pradip Bose Alper Buyuktosunoglu Morgan Kaufmann is an imprint of Elsevier 50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States ISBN: 978-0-12-802459-1

6. Security Analysis and Improvements of Two-Factor Mutual Authentication with Key Agreement in Wireless Sensor Network sensors ISSN 1424-8220

ДОДАТОК 1

Інструментальні засоби посиленого захисту закритих ресурсів

Специфікація

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ4228_20Б

Аркушів 1

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ІМ. ІГОРЯ СІКОРСЬКОГО»_ТЕФ_АПЕПС_ТМ4228_20Б-1	HuseinovRN_TM62_Zapiska.docx	Пояснювальна записка
Комплекс		
Компоненти		
УКР.НТУУ «КПІ ІМ. ІГОРЯ СІКОРСЬКОГО»_ТЕФ_АПЕПС_ТМ4228_20Б-2	dod_3.docx	Опис програмного модуля
УКР.НТУУ «КПІ ІМ. ІГОРЯ СІКОРСЬКОГО»_ТЕФ_АПЕПС_ТМ4228_20Б-3	tokenGen.js	Файл з логікою генерації хеша
УКР.НТУУ «КПІ ІМ. ІГОРЯ СІКОРСЬКОГО»_ТЕФ_АПЕПС_ТМ4228_20Б-4	tokens.js	Mongoose модель колекції токенів
УКР.НТУУ «КПІ ІМ. ІГОРЯ СІКОРСЬКОГО»_ТЕФ_АПЕПС_ТМ4228_20Б-5	token.js	Express контроллер

ДОДАТОК 2

Інструментальні засоби посиленого захисту закритих ресурсів

Текст програмного модуля

УКР.НТУУ “КПІ”.ТМ4228_20Б-1

Аркушів 19

2020

```
(function () {
    var Hashes;

    function utf8Encode(str) {
      var x, y, output = '',
        i = -1,
        l;

      if (str && str.length) {
        l = str.length;
        while ((i += 1) < l) {
          /* Decode utf-16 surrogate pairs */
          x = str.charCodeAt(i);
          y = i + 1 < l ? str.charCodeAt(i + 1) : 0;
          if (0xD800 <= x && x <= 0xDBFF && 0xDC00 <= y && y <= 0xDFFF) {
            x = 0x10000 + ((x & 0x03FF) << 10) + (y & 0x03FF);
            i += 1;
          }
          /* Encode output as utf-8 */
          if (x <= 0x7F) {
            output += String.fromCharCode(x);
          } else if (x <= 0x7FF) {
            output += String.fromCharCode(0xC0 | ((x >>> 6) & 0x1F),
              0x80 | (x & 0x3F));
          } else if (x <= 0xFFFF) {
            output += String.fromCharCode(0xE0 | ((x >>> 12) & 0x0F),
              0x80 | ((x >>> 6) & 0x3F),
              0x80 | (x & 0x3F));
          } else if (x <= 0x1FFFFF) {
            output += String.fromCharCode(0xF0 | ((x >>> 18) & 0x07),
              0x80 | ((x >>> 12) & 0x3F),
              0x80 | ((x >>> 6) & 0x3F),
              0x80 | (x & 0x3F));
          }
        }
      }
      return output;
    }

    function utf8Decode(str) {
      var i, ac, c1, c2, c3, arr = [],
        l;
      i = ac = c1 = c2 = c3 = 0;

      if (str && str.length) {
        l = str.length;
        str += '';

        while (i < l) {
          c1 = str.charCodeAt(i);
          ac += 1;
          if (c1 < 128) {
            arr[ac] = String.fromCharCode(c1);
            i += 1;
          } else if (c1 > 191 && c1 < 224) {
            c2 = str.charCodeAt(i + 1);
            arr[ac] = String.fromCharCode(((c1 & 31) << 6) | (c2 & 63));
            i += 2;
          } else {
            c2 = str.charCodeAt(i + 1);
            c3 = str.charCodeAt(i + 2);
            arr[ac] = String.fromCharCode(((c1 & 15) << 12) | ((c2 & 63)
```

```
<< 6) | (c3 & 63));
            i += 3;
          }
        }
      }
      return arr.join('');
    }

    /**
     * Add integers, wrapping at 2^32. This uses 16-bit operations
internally
     * to work around bugs in some JS interpreters.
     */

    function safe_add(x, y) {
      var lsw = (x & 0xFFFF) + (y & 0xFFFF),
        msw = (x >> 16) + (y >> 16) + (lsw >> 16);
      return (msw << 16) | (lsw & 0xFFFF);
    }

    /**
     * Bitwise rotate a 32-bit number to the left.
     */

    function bit_rol(num, cnt) {
      return (num << cnt) | (num >>> (32 - cnt));
    }

    /**
     * Convert a raw string to a hex string
     */

    function rstr2hex(input, hexcase) {
      var hex_tab = hexcase ? '0123456789ABCDEF' : '0123456789abcdef',
        output = '',
        x, i = 0,
        l = input.length;
      for (; i < l; i += 1) {
        x = input.charCodeAt(i);
        output += hex_tab.charAt((x >>> 4) & 0x0F) + hex_tab.charAt(x &
0x0F);
      }
      return output;
    }

    /**
     * Encode a string as utf-16
     */

    function str2rstr_utf16le(input) {
      var i, l = input.length,
        output = '';
      for (i = 0; i < l; i += 1) {
        output += String.fromCharCode(input.charCodeAt(i) & 0xFF,
(input.charCodeAt(i) >>> 8) & 0xFF);
      }
      return output;
    }

    function str2rstr_utf16be(input) {
      var i, l = input.length,
        output = '';
      for (i = 0; i < l; i += 1) {
        output += String.fromCharCode((input.charCodeAt(i) >>> 8) & 0xFF,
input.charCodeAt(i) & 0xFF);
```

```
    }
    return output;
  }

  /**
   * Convert an array of big-endian words to a string
   */

  function binb2rstr(input) {
    var i, l = input.length * 32,
      output = '';
    for (i = 0; i < l; i += 8) {
      output += String.fromCharCode((input[i >> 5] >>> (24 - i % 32)) &
0xFF);
    }
    return output;
  }

  /**
   * Convert an array of little-endian words to a string
   */

  function binl2rstr(input) {
    var i, l = input.length * 32,
      output = '';
    for (i = 0; i < l; i += 8) {
      output += String.fromCharCode((input[i >> 5] >>> (i % 32)) &
0xFF);
    }
    return output;
  }

  /**
   * Convert a raw string to an array of little-endian words
   * Characters >255 have their high-byte silently ignored.
   */

  function rstr2binl(input) {
    var i, l = input.length * 8,
      output = Array(input.length >> 2),
      lo = output.length;
    for (i = 0; i < lo; i += 1) {
      output[i] = 0;
    }
    for (i = 0; i < l; i += 8) {
      output[i >> 5] |= (input.charCodeAt(i / 8) & 0xFF) << (i % 32);
    }
    return output;
  }

  function rstr2binb(input) {
    var i, l = input.length * 8,
      output = Array(input.length >> 2),
      lo = output.length;
    for (i = 0; i < lo; i += 1) {
      output[i] = 0;
    }
    for (i = 0; i < l; i += 8) {
      output[i >> 5] |= (input.charCodeAt(i / 8) & 0xFF) << (24 - i %
32);
    }
    return output;
  }

  /**
```

```
     * Convert a raw string to an arbitrary string encoding
     */

    function rstr2any(input, encoding) {
      var divisor = encoding.length,
        remainders = Array(),
        i, q, x, ld, quotient, dividend, output, full_length;

      /* Convert to an array of 16-bit big-endian values, forming the
dividend */
      dividend = Array(Math.ceil(input.length / 2));
      ld = dividend.length;
      for (i = 0; i < ld; i += 1) {
        dividend[i] = (input.charCodeAt(i * 2) << 8) | input.charCodeAt(i
* 2 + 1);
      }

      /**
       * Repeatedly perform a long division. The binary array forms the
dividend,
       * the length of the encoding is the divisor. Once computed, the
quotient
       * forms the dividend for the next step. We stop when the dividend
is zerHashes.
       * All remainders are stored for later use.
       */
      while (dividend.length > 0) {
        quotient = Array();
        x = 0;
        for (i = 0; i < dividend.length; i += 1) {
          x = (x << 16) + dividend[i];
          q = Math.floor(x / divisor);
          x -= q * divisor;
          if (quotient.length > 0 || q > 0) {
            quotient[quotient.length] = q;
          }
        }
        remainders[remainders.length] = x;
        dividend = quotient;
      }

      /* Convert the remainders to the output string */
      output = '';
      for (i = remainders.length - 1; i >= 0; i--) {
        output += encoding.charAt(remainders[i]);
      }

      /* Append leading zero equivalents */
      full_length = Math.ceil(input.length * 8 /
(Math.log(encoding.length) / Math.log(2)));
      for (i = output.length; i < full_length; i += 1) {
        output = encoding[0] + output;
      }
      return output;
    }

    /**
     * Convert a raw string to a base-64 string
     */

    function rstr2b64(input, b64pad) {
      var tab =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/',
        output = '',
        len = input.length,
```

```
        i, j, triplet;
      b64pad = b64pad || '=';
      for (i = 0; i < len; i += 3) {
        triplet = (input.charCodeAt(i) << 16) | (i + 1 < len ?
input.charCodeAt(i + 1) << 8 : 0) | (i + 2 < len ? input.charCodeAt(i +
2) : 0);
        for (j = 0; j < 4; j += 1) {
          if (i * 8 + j * 6 > input.length * 8) {
            output += b64pad;
          } else {
            output += tab.charAt((triplet >>> 6 * (3 - j)) & 0x3F);
          }
        }
      }
      return output;
    }

    Hashes = {
      /**
       * @property {String} version
       * @readonly
       */
      VERSION: '1.0.6',
      /**
       * @member Hashes
       * @class Base64
       * @constructor
       */

      /**
       * @class Hashes.SHA256
       * @param {config}
       *
       * A JavaScript implementation of the Secure Hash Algorithm, SHA-
256, as defined in FIPS 180-2
       * Version 2.2 Copyright Angel Marin, Paul Johnston 2000 - 2009.
       * Other contributors: Greg Holt, Andrew Kepert, Ydnar, Lostinet
       * See http://pajhome.org.uk/crypt/md5 for details.
       * Also http://anmar.eu.org/projects/jssha2/
       */
      SHA256: function (options) {
        /**
         * Private properties configuration variables. You may need to
tweak these to be compatible with
         * the server-side, but the defaults work in most cases.
         * @see this.setUpperCase() method
         * @see this.setPad() method
         */
        var hexcase = (options && typeof options.uppercase === 'boolean')
? options.uppercase : false, // hexadecimal output case format. false -
lowercase; true - uppercase  */
          b64pad = (options && typeof options.pad === 'string') ?
options.pad : '=',
          /* base-64 pad character. Default '=' for strict RFC compliance
*/
          utf8 = (options && typeof options.utf8 === 'boolean') ?
options.utf8 : true,
          /* enable/disable utf8 encoding */
          sha256_K;

        /* privileged (public) methods */
        this.hex = function (s) {
          return rstr2hex(rstr(s, utf8));
        };
        this.b64 = function (s) {
```

```
          return rstr2b64(rstr(s, utf8), b64pad);
        };
        this.any = function (s, e) {
          return rstr2any(rstr(s, utf8), e);
        };
        this.raw = function (s) {
          return rstr(s, utf8);
        };
        this.hex_hmac = function (k, d) {
          return rstr2hex(rstr_hmac(k, d));
        };
        this.b64_hmac = function (k, d) {
          return rstr2b64(rstr_hmac(k, d), b64pad);
        };
        this.any_hmac = function (k, d, e) {
          return rstr2any(rstr_hmac(k, d), e);
        };
        /**
         * Perform a simple self-test to see if the VM is working
         * @return {String} Hexadecimal hash sample
         * @public
         */
        this.vm_test = function () {
          return hex('abc').toLowerCase() ===
'900150983cd24fb0d6963f7d28e17f72';
        };
        /**
         * Enable/disable uppercase hexadecimal returned string
         * @param {boolean}
         * @return {Object} this
         * @public
         */
        this.setUpperCase = function (a) {
          if (typeof a === 'boolean') {
            hexcase = a;
          }
          return this;
        };
        /**
         * @description Defines a base64 pad string
         * @param {string} Pad
         * @return {Object} this
         * @public
         */
        this.setPad = function (a) {
          b64pad = a || b64pad;
          return this;
        };
        /**
         * Defines a base64 pad string
         * @param {boolean}
         * @return {Object} this
         * @public
         */
        this.setUTF8 = function (a) {
          if (typeof a === 'boolean') {
            utf8 = a;
          }
          return this;
        };

        // private methods

        /**
         * Calculate the SHA-512 of a raw string
```

```
         */

        function rstr(s, utf8) {
          s = (utf8) ? utf8Encode(s) : s;
          return binb2rstr(binb(rstr2binb(s), s.length * 8));
        }

        var subTable = [

        ];

        var subTable3 = [ 5, 3, 9, 12, 9, 10, 3, 9, 6, 12, 5, 6, 3, 6,
12, 10 ]

        var subTable2 = [
          [0, 9, 6, 3]
          [3, 10, 12, 6],
          [9, 3, 5, 12],
          [12, 9, 6, 10]
        ];

        /**
         * Calculate the HMAC-sha256 of a key and some data (raw strings)
         */

        function rstr_hmac(key, data) {
          key = (utf8) ? utf8Encode(key) : key;
          data = (utf8) ? utf8Encode(data) : data;
          var hash, i = 0,
            bkey = rstr2binb(key),
            ipad = Array(16),
            opad = Array(16);

          var messageHash = binb(ipad.concat(rstr2binb(data)), 512 +
data.length * 8);
          var res = [];


          for (var j = 0; j < messageHash.length; j++) {
            res.push(messageHash[j] ^ bkey[j]);
          }

          var g = 0;

          var A = 0,
            B = 0;
          for (var k = 0; k < res.length; k++) {
            g++;
            var x = 0,
              y = 0;

            for (var l = 0; l < 3; l++) {
              x = x << 1 | (res[(g*k+l) % res.length] & 1);
            }

            y = ~x&15;

            var s = subTable3[x];
            var v = subTable3[y];

            A = A << 4 | (s & 15);
            B = B << 4 | (v & 15);
          }
```

```
  if (bkey.length > 16) {
    bkey = binb(bkey, key.length * 8);
  }

  for (; i < 16; i += 1) {
    ipad[i] = bkey[i] ^ A;
    opad[i] = bkey[i] ^ B;
  }


  hash = binb(ipad.concat(rstr2binb(data)), 512 + data.length *
8);
  return binb2rstr(binb(opad.concat(hash), 512 + 256));
}

/*
 * Main sha256 function, with its support functions
 */


function sha256_S(X, n) {
  return (X >>> n) | (X << (32 - n));
}

function sha256_R(X, n) {
  return (X >>> n);
}

function sha256_Ch(x, y, z) {
  return ((x & y) ^ ((~x) & z));
}

function sha256_Maj(x, y, z) {
  return ((x & y) ^ (x & z) ^ (y & z));
}

function sha256_Sigma0256(x) {
  return (sha256_S(x, 2) ^ sha256_S(x, 13) ^ sha256_S(x, 22));
}

function sha256_Sigma1256(x) {
  return (sha256_S(x, 6) ^ sha256_S(x, 11) ^ sha256_S(x, 25));
}

function sha256_Gamma0256(x) {
  return (sha256_S(x, 7) ^ sha256_S(x, 18) ^ sha256_R(x, 3));
}

function sha256_Gamma1256(x) {
  return (sha256_S(x, 17) ^ sha256_S(x, 19) ^ sha256_R(x, 10));
}

function sha256_Sigma0512(x) {
  return (sha256_S(x, 28) ^ sha256_S(x, 34) ^ sha256_S(x, 39));
}

function sha256_Sigma1512(x) {
  return (sha256_S(x, 14) ^ sha256_S(x, 18) ^ sha256_S(x, 41));
}

function sha256_Gamma0512(x) {
  return (sha256_S(x, 1) ^ sha256_S(x, 8) ^ sha256_R(x, 7));
}

function sha256_Gamma1512(x) {
```

```
          return (sha256_S(x, 19) ^ sha256_S(x, 61) ^ sha256_R(x, 6));
        }

        sha256_K = [
          1116352408, 1899447441, -1245643825, -373957723, 961987163,
1508970993, -1841331548, -1424204075, -670586216, 310598401, 607225278,
1426881987,
          1925078388, -2132889090, -1680079193, -1046744716, -459576895,
-272742522,
          264347078, 604807628, 770255983, 1249150122, 1555081692,
1996064986, -1740746414, -1473132947, -1341970488, -1084653625, -
958395405, -710438585,
          113926993, 338241895, 666307205, 773529912, 1294757372,
1396182291,
          1695183700, 1986661051, -2117940946, -1838011259, -1564481375,
-1474664885, -1035236496, -949202525, -778901479, -694614492, -200395387,
275423344,
          430227734, 506948616, 659060556, 883997877, 958139571,
1322822218,
          1537002063, 1747873779, 1955562222, 2024104815, -2067236844, -
1933114872, -1866530822, -1538233109, -1090935817, -965641998
        ];

        function binb(m, l) {
          var HASH = [1779033703, -1150833019, 1013904242, -1521486534,
            1359893119, -1694144372, 528734635, 1541459225
          ];
          var W = new Array(64);
          var a, b, c, d, e, f, g, h;
          var i, j, T1, T2;

          /* append padding */
          m[l >> 5] |= 0x80 << (24 - l % 32);
          m[((l + 64 >> 9) << 4) + 15] = l;

          for (i = 0; i < m.length; i += 16) {
            a = HASH[0];
            b = HASH[1];
            c = HASH[2];
            d = HASH[3];
            e = HASH[4];
            f = HASH[5];
            g = HASH[6];
            h = HASH[7];

            for (j = 0; j < 64; j += 1) {
              if (j < 16) {
                W[j] = m[j + i];
              } else {
                W[j] = safe_add(safe_add(safe_add(sha256_Gamma1256(W[j -
2]), W[j - 7]),
                  sha256_Gamma0256(W[j - 15])), W[j - 16]);
              }

              T1 = safe_add(safe_add(safe_add(safe_add(h,
sha256_Sigma1256(e)), sha256_Ch(e, f, g)),
                sha256_K[j]), W[j]);
              T2 = safe_add(sha256_Sigma0256(a), sha256_Maj(a, b, c));
              h = g;
              g = f;
              f = e;
              e = safe_add(d, T1);
              d = c;
              c = b;
              b = a;
```

```
            a = safe_add(T1, T2);
          }

          HASH[0] = safe_add(a, HASH[0]);
          HASH[1] = safe_add(b, HASH[1]);
          HASH[2] = safe_add(c, HASH[2]);
          HASH[3] = safe_add(d, HASH[3]);
          HASH[4] = safe_add(e, HASH[4]);
          HASH[5] = safe_add(f, HASH[5]);
          HASH[6] = safe_add(g, HASH[6]);
          HASH[7] = safe_add(h, HASH[7]);
        }
        return HASH;
      }

    },



  };

  // exposes Hashes
  (function (window, undefined) {
    var freeExports = false;
    if (typeof exports === 'object') {
      freeExports = exports;
      if (exports && typeof global === 'object' && global && global ===
global.global) {
        window = global;
      }
    }

    if (typeof define === 'function' && typeof define.amd === 'object'
&& define.amd) {
      // define as an anonymous module, so, through path mapping, it
can be aliased
      define(function () {
        return Hashes;
      });
    } else if (freeExports) {
      // in Node.js or RingoJS v0.8.0+
      if (typeof module === 'object' && module && module.exports ===
freeExports) {
        module.exports = Hashes;
      }
      // in Narwhal or RingoJS v0.7.0-
      else {
        freeExports.Hashes = Hashes;
      }
    } else {
      // in a browser or Rhino
      window.Hashes = Hashes;
    }
  }(this));
}()); // IIFE
```



```
var mongoose = require( 'mongoose' );
var crypto = require('crypto');
var uid = require("uid");
var tokenGen = require('../tokenGen');
var shaHmac = new tokenGen.SHA256;
```

```
var tokenSchema = new mongoose.Schema({
  secretKey: String,
  userId: String,
  valid: Boolean
});

tokenSchema.methods.generateSecret = function () {
  this.secretKey = uid(10);
}

tokenSchema.methods.verifyToken = function (token) {
  var time = Math.round((Date.now() - Date.now() % (45*1000))/1000) *
1000;

  time = time + '';


  var genToken = shaHmac.b64_hmac(this.secretKey, time);

  var numberToken = Number.parseInt(genToken.substring(0,
6).split('').map(s => s.charCodeAt(0) % 10).join(''), 10);
```



```
  console.log(`Time is : ${time}`);
  console.log(`Secret key is : ${this.secretKey}`);
  console.log(`Token is : ${numberToken} received ${token}`);

  return numberToken == token;
}

mongoose.model('Token', tokenSchema);

var mongoose = require('mongoose');
var Token = mongoose.model('Token');

module.exports.generateToken = function (req, res) {

  var token = new Token();

  token.userId = req.body.userId;
  token.generateSecret();

  token.save(function (err) {
    res.status(200);
    res.json({
      "secretKey": token.secretKey
    });
  });

};

module.exports.validateToken = function (req, res) {
  var userId = req.body.userId;
  var reqToken = req.body.token;

  Token.findOne({ userId: userId })
    .exec(function (err, token) {
      if (token.verifyToken(reqToken)) {
        token.valid = true;
        token.save(function (err) {
          if (err) {
```

```
            res.status(401);
            res.json({
              success: false
            })
          } else {
            res.json({
              success: true
            });
          }
        });
      } else {
        res.status(401);
        res.json({
            success: false
        })
      }
    });
};

module.exports.submitToken = function (req, res) {
  var userId = req.body.userId;
  var reqToken = req.body.token;

  Token.findOne({ userId: userId })
    .exec(function (err, token) {
      if (token.verifyToken(reqToken)) {
        res.json({
          success: true
        });
      } else {
        res.status(401);
        res.json({
          success: false
        })
      }

    });
}
```

ДОДАТОК З

Інструментальні засоби посиленого захисту закритих ресурсів

Опис програмного модуля

УКР.НТУУ «КПІ ІМ. ІГОРЯ СІКОРСЬКОГО»_ТЕФ_АПЕПС_ТМ4228_20Б-2

Аркушів 7

2020

АНОТАЦІЯ

Розроблене програмне забезпечення призначене для організації посиленого захисту закритих ресурсів.

Результатом виконання є клієнт-серверний застосунок, що організовує систему двохфактроної авторизації.

Мова програмної реалізації — JavaScript з використанням express, Angular та Mongoose фреймворків. Основне середовище розробки — Visual Studio Code. Додаткові програмні застосунки — MongoDB.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Найменування програмного модуля — “Сервер Автентифікації”. Мова програмної реалізації — JavaScript. Середовище розробки — Visual Studio Code.

Даний модуль являє серверний застосунок, який слугує для видачі секретних ключів і валідації одноразових паролів.

Для використання програмного забезпечення у якості клієнта необхідно :

— комп’ютер, чи смартфон з довільною ОС;

— будь-який браузер або http клієнт.

Для використання програмного забезпечення у якості серверу необхідно :

— комп’ютер з довільною ОС;

— встановлена програмна плафторма nodejs;

— розгорнута mongodb база даних.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний модуль надає системі можливість видавати секретні ключі кінцевим користувачам і надавати клієнтській системі механізм для перевірки одноразових паролів, які буде згенеровано на стороні користувачів.

Основним його призначенням можна вважати створення наступних файлів:

— tokenGen.js;

— tokens.js;

— token.js.

Основний функціонал програмного забезпечення, який відповідає за надання веб API є файл token.js:

— generateToken(req, res) — відповідає за генерацію секретного ключа для користувача.

— validateToken(req, res) — відповідає за початкове підтвердження користувачем отримання секретного ключа і завершення його реєстрації.

— submitToken(req, res) — відповідає за валідацію одноразового паролю.

3. ВХІДНІ ДАНІ

Вхідними даними для роботи програмної системи є ідентифікатор користувача, а також одноразовий пароль який був згенерований в додатку.

Вихідними даними є:

— Секретний ключ.

— Результат валідації одноразового паролю.

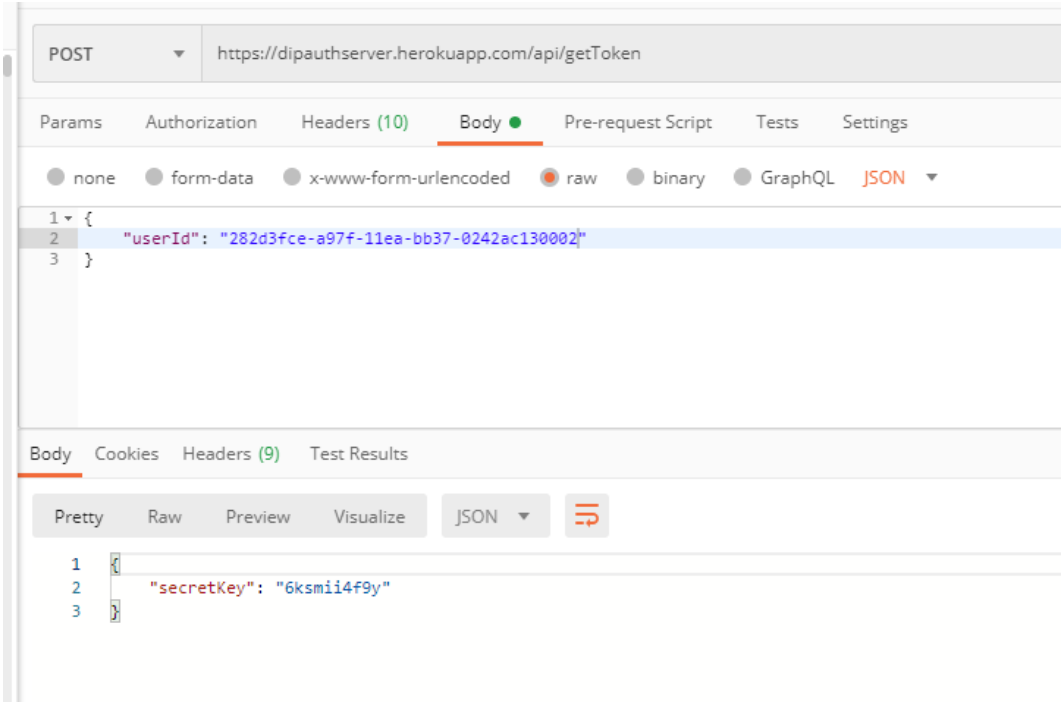

Рисунок 3.1 — Запит на отримання секретного ключа

4. ВИХІДНІ ДАНІ

Вихідними даними результату роботи програмної системи є веб-сторінка, на якій для користувача зображено секретний ключ у вигляді QR-коду.

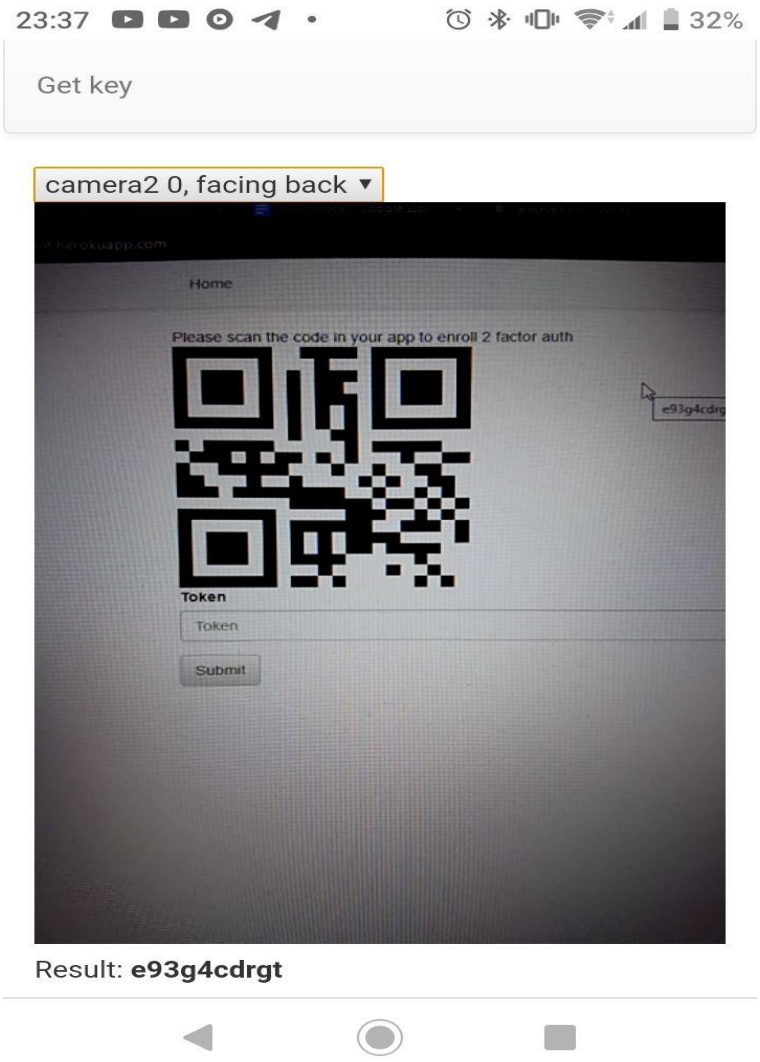

Рисунок 4.1 — Вікно з секретним ключем у вигляді QR-коду